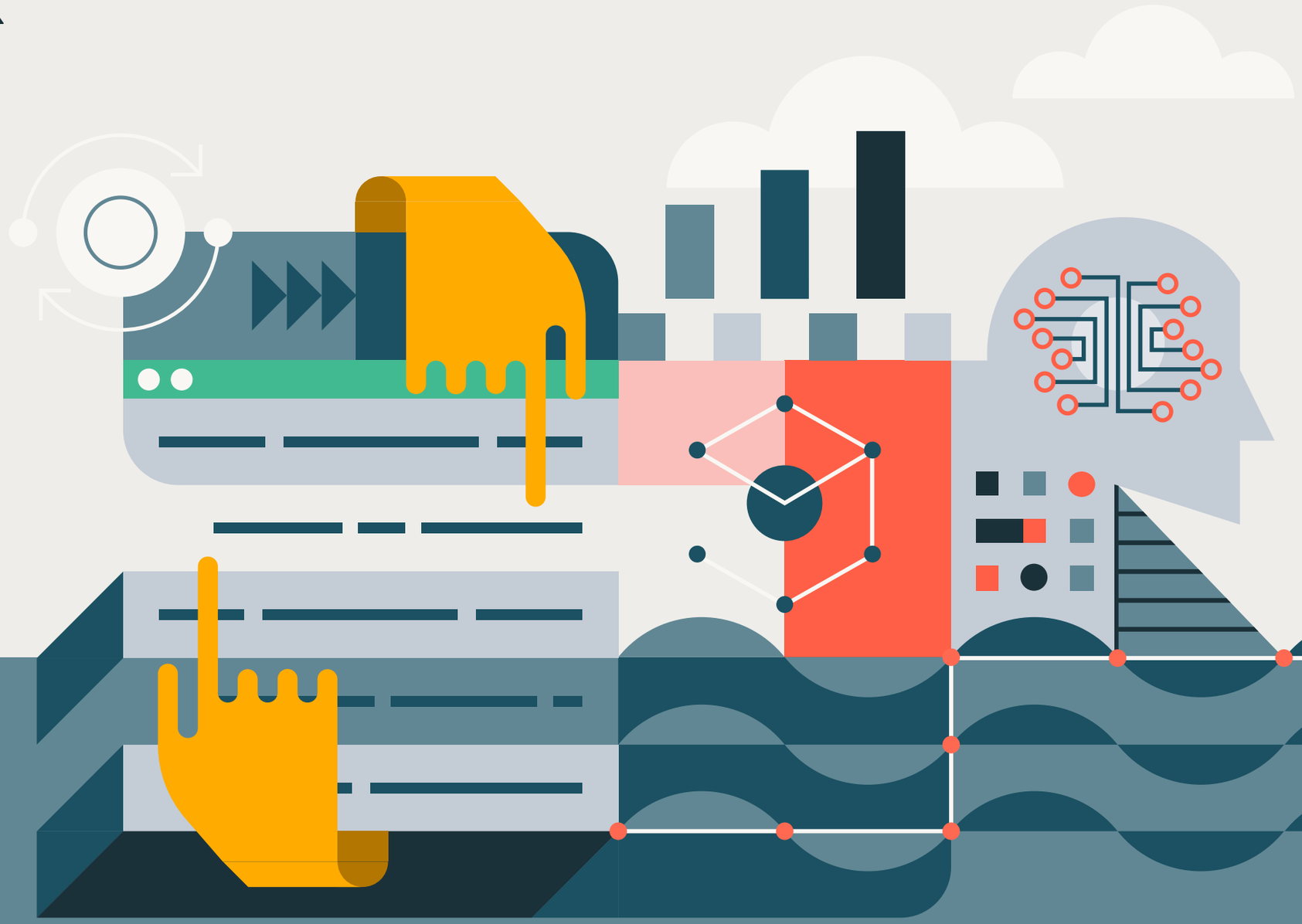
eBook

머신 러닝 사용 사례 Big Book

코드 샘플, 노트북을 비롯한
기술 블로그 컬렉션



databricks

목차

1장:

소개

머신 러닝의 세계는 발전 속도가 너무나 빨라서 현재 개발하고자 하는 주제와 관련된 실제 사용 사례를 찾기가 어렵습니다. 그래서 산업 분야의 선구적 이론가들이 운영하는 블로그에서 지금 바로 사용할 수 있는 실용적 사용 사례를 모았습니다. 이 입문 참조 가이드는 Databricks 플랫폼에서 직접 머신 러닝을 구현해볼 수 있도록 코드 샘플을 포함하여 모든 필수 정보를 제공합니다.

2장:

동적 시간 왜곡의 이해

1부 동적 시간 왜곡 및 MLflow를 사용한 매출 동향 발견 시리즈

글: Ricardo Portilla, Brenner Heintz 및 Denny Lee

2019년 4월 30일

Databricks에서 노트북 보기 →

소개

"동적 시간 왜곡(dynamic time warping)"이라는 표현을 처음 읽으면 〈백투더퓨처〉 시리즈에서 주인공 마티 맥플라이가 드로리언을 시속 142km로 달리던 모습을 떠올릴 수도 있습니다. 아, 동적 시간 왜곡은 시간 여행과는 관계가 없습니다. 비교 데이터 포인트 사이의 시간 색인이 완전히 동기화되지 않을 때 동적으로 시계열을 비교하는 데 사용하는 기술입니다.

나중에 설명해 드리겠지만 동적 시간 왜곡의 가장 핵심적인 용도는 음성 인식입니다. 어떤 구절을 빠르거나 느리게 말하더라도 다른 구절과 일치하는지 알아내는 데 사용합니다. Google Home이나 Amazon Alexa 기기를 활성화시키는 "깨우는 말(wake words)"을 인식할 때 얼마나 유용할지 상상이 가시죠? 모닝커피를 마시지 못해서 다소 말이 어눌하게 나오더라도 인식할 수 있습니다.

동적 시간 왜곡은 다양한 분야에 적용할 수 있는 유용하고 효과적인 기술입니다. 동적 시간 왜곡의 개념을 이해했다면 일상생활에서 활용되는 사례를 쉽게 발견할 수 있고, 앞으로도 기대가 큰 응용 분야입니다. 다음과 같은 사례를 생각해볼 수 있습니다.

- **금융 시장:** 비슷한 기간에 대해 완벽하게 일치하지 않는 증권 거래 데이터를 비교합니다. 예를 들어 2월(28일)과 3월(31일)에 대한 월간 거래 데이터를 비교하는 것입니다.
- **웨어러블 피트니스 트래커:** 사용자가 걷는 속도가 시간에 따라 달라지더라도 걷는 속도와 걸음 수를 정확히 계산합니다.
- **경로 계산:** 운전 습관에 대해 알고 있을 경우(예: 직진 도로에서는 빠르게 운전하지만 좌회전할 때는 평균보다 시간이 더 걸리는 습관) 운전자의 ETA에 대해 보다 정확한 정보를 계산합니다.

데이터 사이언티스트, 데이터 분석가, 시계열을 사용해서 일하는 사람이라면 누구나 이 기술을 알고 있어야 합니다. 완벽하게 일치하는 시계열 비교 데이터는 실전에서 완벽하게 "깔끔한" 데이터만큼이나 보기 어렵기 때문입니다.

블로그 시리즈에서 다룰 내용:

- 동적 시간 왜곡의 기본 원칙
- 샘플 오디오 데이터에서 동적 시간 왜곡 실행
- MLflow를 사용하여 샘플 매출 데이터에서 동적 시간 왜곡 실행

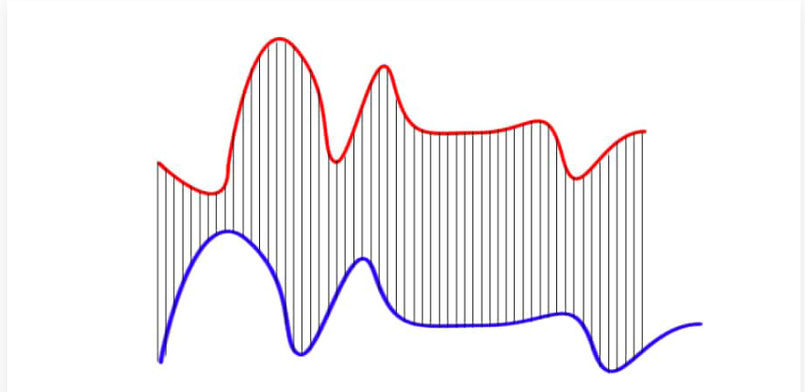
동적 시간 왜곡

시계열 비교법의 목표는 두 입력 시계열 사이에 *거리 척도*를 만드는 것입니다. 일반적으로 두 시계열의 유사성이나 비유사성은 데이터를 벡터로 변환하여 벡터 공간에서 두 지점 사이의 유클리드 거리(직선거리)를 계산하여 산출합니다.

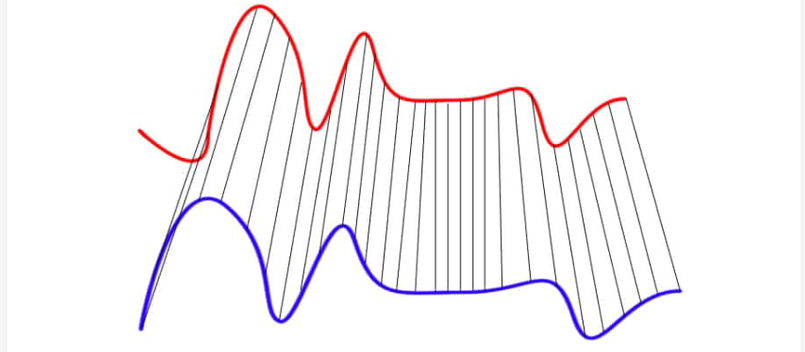
동적 시간 왜곡은 1970년대에 음파 소스에서 음성과 단어를 인식하는 데 사용하던 획기적인 시계열 비교 기법입니다. 주로 인용되는 논문은 "Dynamic time warping for isolated word recognition based on ordered graph searching techniques입니다."

배경

이 기술은 패턴 매칭뿐만 아니라, 이상 탐지에도 사용할 수 있습니다(예: 두 개의 연속적이지 않은 기간 사이의 시계열을 중첩해서 그 형태가 크게 변화하는지 확인하거나 이상치를 찾습니다). 예를 들어, 이 그래프에서 빨간색과 파란색 선을 보면 기존 시계열 매칭(즉, 유클리드 매칭)은 매우 제한적입니다. 반면, 동적 시간 왜곡을 사용하면 X축(즉, 시간)이 동기화되지 않았더라도 두 곡선이 균등하게 매칭됩니다. 이 방법은 로버스트 상이도 점수로 생각해도 됩니다. 숫자가 낮을수록 시계열이 더욱 유사한 것을 의미합니다.



유클리드 매칭



동적 시간 왜곡 매칭

출처: Wikimedia Commons
File: Euclidean_vs_DTW.jpg

두 시계열(기준 시계열 및 새로운 시계열)은 다음의 규칙을 따르는 함수 f(x)로 매핑해서 최적(왜곡) 경로를 사용하여 크기를 매칭할 수 있을 때 유사한 것으로 간주합니다.

$$f(x_i) \; maps \; to \; f(x_j) \; when \; i <= j$$

$$f(x_i) \; maps \; to \; f(x_j) \; only \; when \; (j - i) \; is \; within \; fixed \; range$$

사운드 패턴 매칭

원래 동적 시간 왜곡은 오디오 클립에 적용하여 해당 클립의 유사성을 알아내는 데 사용했습니다. 우리 예시에서는 "The Expanse."라는 TV 드라마의 대사가 나오는 오디오 클립 4개를 사용하겠습니다. 오디오 클립은 4개가 있습니다(아래에서 들어볼 수는 있지만 들어보지 않으셔도 됩니다). 그중 3개(클립 1, 2, 4)의 대사는 다음과 같습니다.

"문과 구석진 곳, 그런 데서 당하는 거야."

나머지 1개(클립 3)의 대사는 다음과 같습니다.

"방 안으로 너무 빨리 들어가면, 방에 잡아먹히지."

클립 1 문과 구석진 곳, 그런 데서 당하는 거야. [v1]

0:00 / 0:06

클립 2 문과 구석진 곳, 그런 데서 당하는 거야. [v2]

0:00 / 0:08

클립 3 방 안으로 너무 빨리 들어가면, 방에 잡아먹히지.

0:00 / 0:07

클립 4 문과 구석진 곳, 그런 데서 당하는 거야. [v3]

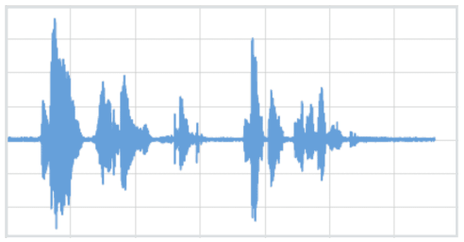
0:00 / 0:07

"The Expanse"의 대사

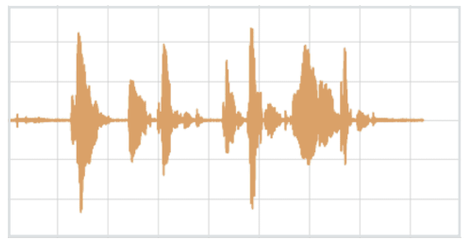
다음은 4가지 오디오 클립을 `matplotlib`으로 시각화한 결과입니다.

- **클립 1:** "문과 구석진 곳, 그런 데서 당하는 거야"라는 대사가 나오는 기본 시계열입니다.
- **클립 2:** 클립 1에 기반한 새로운 시계열[v2]로, 어조와 말하는 패턴이 매우 과장되어 있습니다.
- **클립 3:** 클립 1과 동일한 어조와 음성이지만 "방 안으로 너무 빨리 들어가면, 방에 잡아먹히지"라는 대사가 나오는 시계열입니다.
- **클립 4:** 클립 1에 기반한 새로운 시계열[v3]로, 어조와 말하는 패턴이 클립 1과 유사합니다.

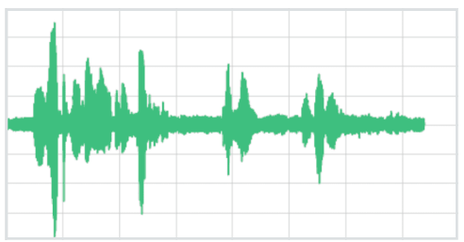
클립 1 문과 구석진 곳, 그런 데서 당하는 거야. [v1]



클립 2 문과 구석진 곳, 그런 데서 당하는 거야. [v2]



클립 3 방 안으로 너무 빨리 들어가면, 방에 잡아먹히지.



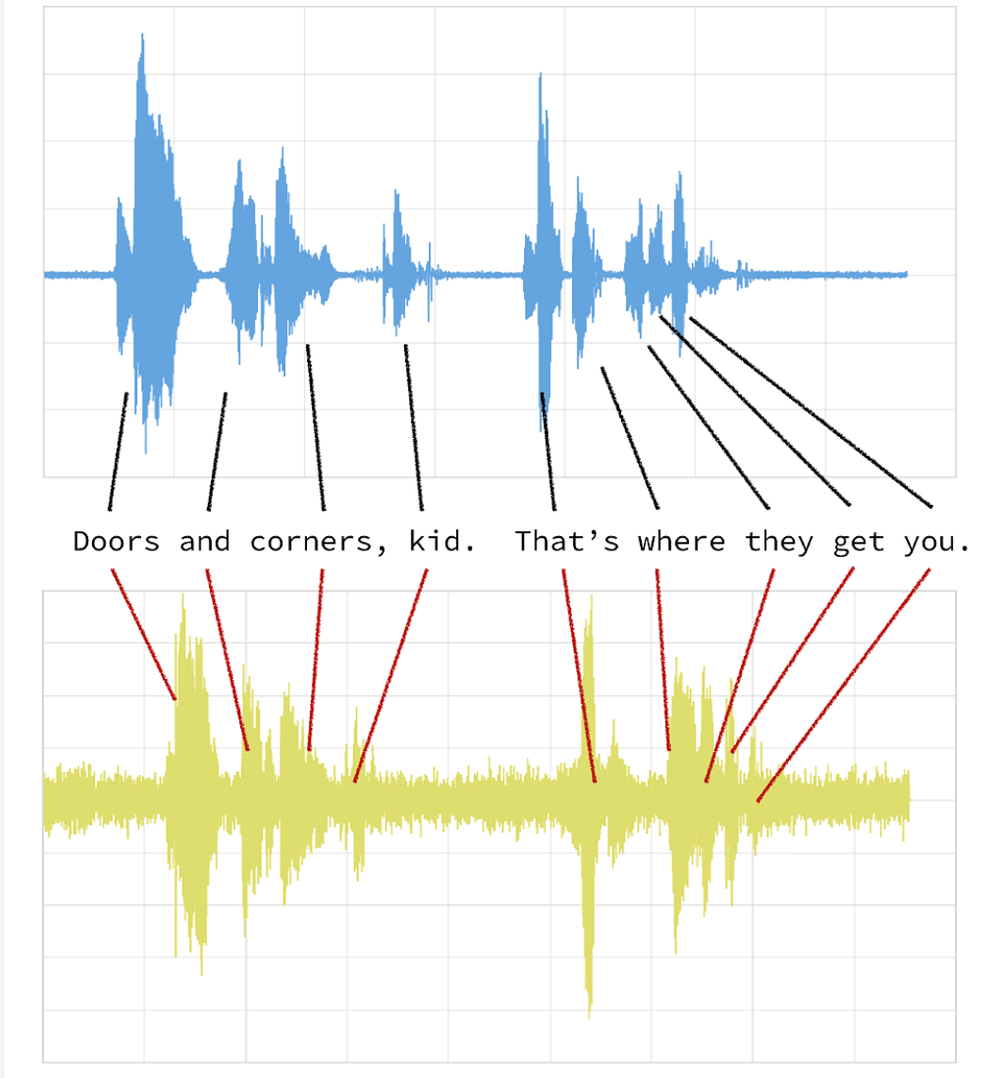
클립 4 문과 구석진 곳, 그런 데서 당하는 거야. [v3]

오디오 클립을 읽고 matplotlib을 사용하여 시각화하는 코드는 다음의 코드 조각으로 요약할 수 있습니다.

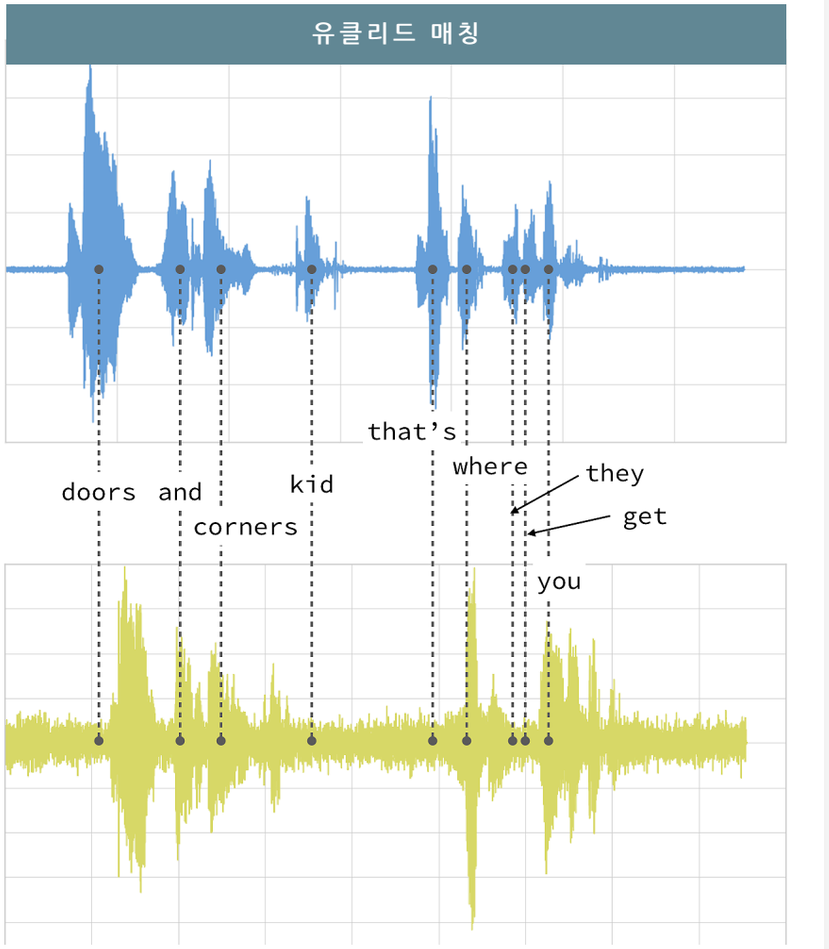
```
from scipy.io import wavfile
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure

# Read stored audio files for comparison
fs, data = wavfile.read("/dbfs/folder/clip1.wav")

# Set plot style
plt.style.use('seaborn-whitegrid')

# Create subplots
ax = plt.subplot(2, 2, 1)
ax.plot(data1, color='#67A0DA')
...

# Display created figure
fig=plt.show()
display(fig)
```

전체 코드베이스는 **동적 시간 왜곡 배경** 노트북에 나와 있습니다.

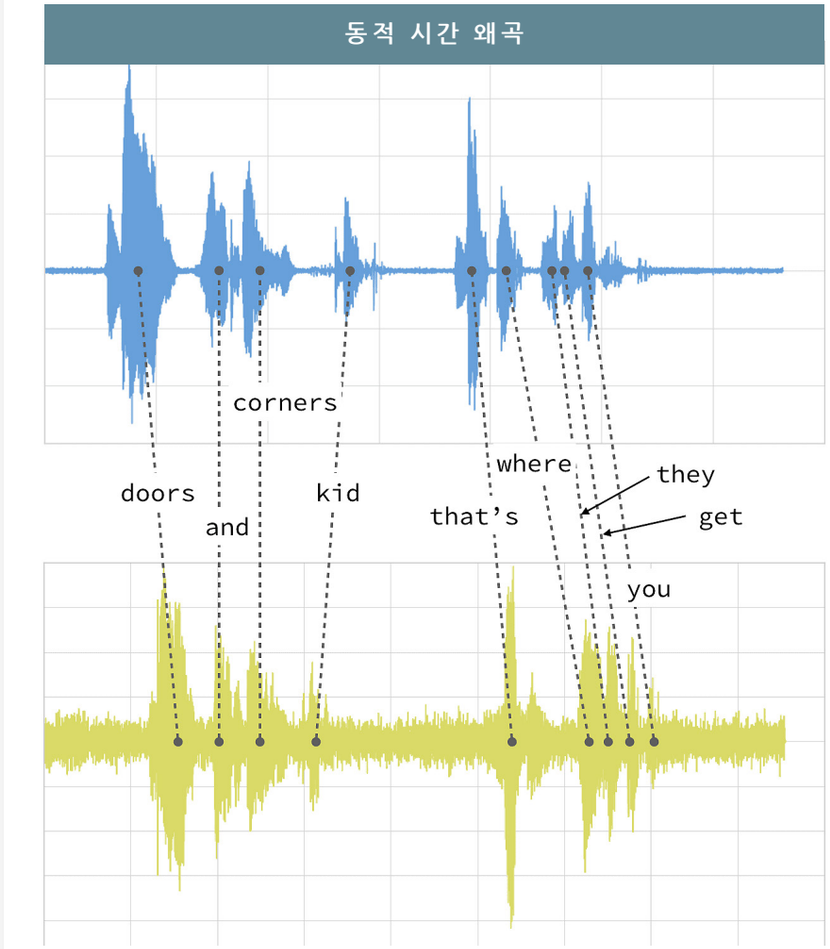
아래에서 설명했듯이, 두 클립(이 경우 클립 1과 4)은 같은 대사지만 어조(진폭)와 지연 시간이 다릅니다.



기존 유클리드 매칭(아래 그래프 참조)을 따를 경우, 진폭을 무시하더라도 원본 클립(파란색)과 새로운 클립(노란색) 간의 타이밍이 일치하지 않습니다.



동적 시간 왜곡을 적용하면 시간을 옮겨서 두 클립의 시계열을 비교할 수 있습니다.

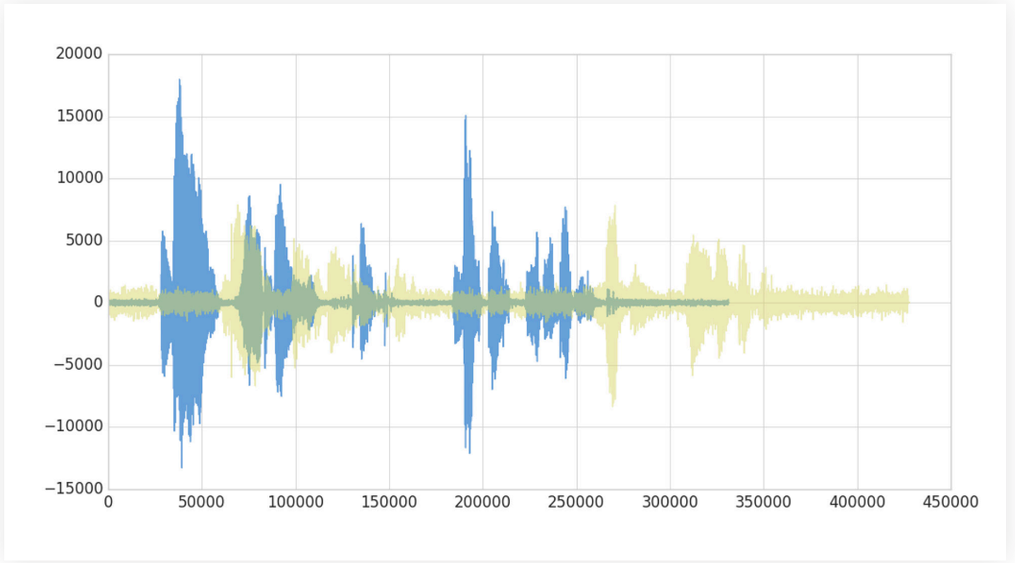


이 시계열 비교에서는 `fastdtw` PyPi 라이브러리를 사용합니다. Databricks 작업 영역에 PyPi 라이브러리를 설치하기 위한 명령은 Azure | AWS에서 확인할 수 있습니다.

```
from fastdtw import fastdtw

# Distance between clip 1 and clip 2
distance = fastdtw(data_clip1, data_clip2)[0]
print("The distance between the two clips is %s" % distance)
```

전체 코드베이스는 **동적 시간 왜곡 배경** 노트북에 나와 있습니다.

기준	쿼리	거리
클립 1	클립 2	480148446.0
	클립 3	310038909.0
	클립 4	293547478.0



관찰 결과 요약:

- 앞서 보여드린 그래프에서 확인하였듯이, 클립 1과 4는 어조와 단어가 동일하므로 가장 거리가 짧습니다.
- 클립 1과 3 사이의 거리도 (클립 4에 비해 길기는 하지만) 상당히 짧습니다. 단어는 다르지만 어조가 동일하고 말하는 속도가 같습니다.
- 클립 1과 2는 같은 대사이기는 하지만 어조와 말하는 속도가 매우 과장되어 있어서 거리가 가장 깁니다.

여러분도 보다시피, 동적 시간 왜곡을 사용하면 두 시계열의 유사성을 알아낼 수 있습니다.

다음

동적 시간 왜곡에 관해 설명해드렸으므로 이 사용 사례를 **매출 동향을 탐지**하는 데 적용해보겠습니다.

2장:

동적 시간 왜곡 및 MLflow를 사용한 매출 동향 발견

2부 동적 시간 왜곡 및 MLflow를 사용한 매출 동향 발견 시리즈

글: Ricardo Portilla, Brenner Heintz 및 Denny Lee

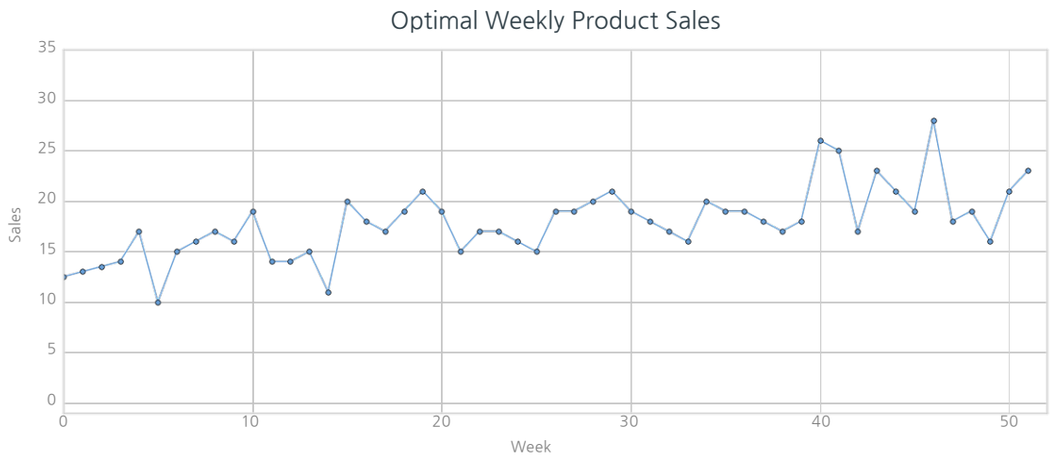
2019년 4월 30일

Databricks에서 이 노트북 시리즈 보기 (DBC 형식) →

배경

여러분이 3D 프린팅 제품을 제작하는 회사를 소유한 사장이라고 생각해보세요. 작년에 드론 프로펠러의 수요가 매우 꾸준하다는 것을 알았기 때문에 그 제품을 판매했습니다. 그 전 해에는 휴대전화 케이스를 판매했습니다. **조만간 새해를 맞이하기에, 제조팀과 내년에 생산할 제품에 대해 협의하고 있습니다.** 3D 프린터를 구매하려면 많은 대출을 받아야 하기 때문에 대출을 상환하려면 구매한 프린터를 항상 100%나 그에 가깝게 사용해야 합니다.

여러분은 현명한 CEO이기 때문에 내년 생산 능력이 변동할 수밖에 없다는 것을 알고 있고 생산 능력이 유난히 높은 주도 있다는 것을 압니다. 예를 들어 여름에 (임시 근로자를 채용하면) 생산 능력이 높아지고, 매월 셋째 주에는 (3D 프린터 필라멘트 공급망 문제로 인해) 생산 능력이 저하됩니다. 아래의 그래프는 회사의 생산 능력 추산치입니다.



databricks

주간 수요가 생산 능력과 최대한 가까운 제품을 선택하는 것이 목표입니다. 각 제품에 대한 작년 매출을 포함한 카탈로그를 살펴보고 있는데, 올해 매출도 비슷할 것으로 예상합니다.

주간 수요가 자신의 생산 능력을 초과하는 제품을 선택할 경우, 고객 주문을 취소해야 하므로 사업에 좋지 못합니다. 반면, 주간 수요가 충분하지 않은 제품을 선택한다면 프린터의 사용률을 최대로 높일 수 없어서 대출을 상환하지 못할 수도 있습니다.

이럴 때 동적 시간 왜곡을 사용합니다. 선택한 제품의 공급과 수요가 다소 일치하지 않을 수 있기 때문입니다. 모든 수요를 감당할 수 있을 만큼 생산 여력이 충분하지 않은 주도 있겠지만, 차이가 지나치게 크지 않다면 그 주보다 한두 주 전후로 제품을 더 많이 생산해서 보충한다면 고객에게는 영향이 미치지 않을 것입니다. 유클리드 매칭을 사용해서 매출 데이터와 생산 능력을 비교하는 데 그친다면, 이를 고려하지 않은 제품을 선택해서 돈을 벌 기회를 놓치게 됩니다. 대신 우리는 동적 시간 왜곡을 사용해서 올해 회사 상황에 맞는 제품을 선택할 것입니다.

제품 매출 데이터 세트 로드

UCI 데이터 세트 리포지토리에 있는 **주간 매출 거래 데이터 세트**를 사용해서 매출 기반 시계열 분석을 적용합니다. (출처: James Tan, **jamestansc@suss.edu.sg**, Singapore University of Social Sciences)

```
import pandas as pd

# Use Pandas to read this data
sales_pdf = pd.read_csv(sales_dbfspath, header='infer')

# Review data
display(spark.createDataFrame(sales_pdf))
```

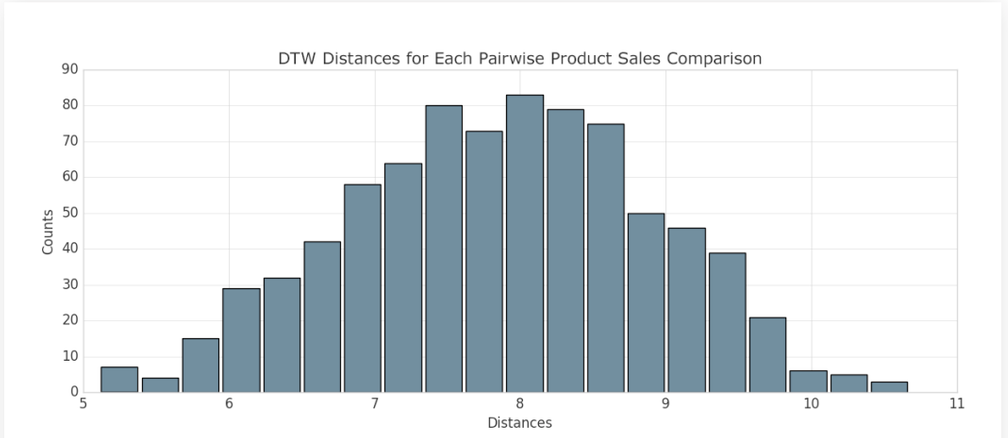
Product_Code	W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13
P1	11	12	10	8	13	12	14	21	6	14	11	14	16	9
P2	7	6	3	2	7	1	6	3	3	3	2	2	6	2
P3	7	11	8	9	10	8	7	13	12	6	14	9	4	7
P4	12	8	13	5	9	6	9	13	13	11	8	4	5	4
P5	8	5	13	11	6	7	9	14	9	9	11	18	8	4
P6	3	3	2	7	6	3	8	6	6	3	1	1	5	4
P7	4	8	3	7	8	7	2	3	10	3	5	2	3	4
P8	8	6	10	9	6	8	7	5	10	10	8	8	15	9

각 제품은 행으로 나타내고 올해의 각 주는 열로 나타냅니다. 값은 매주 판매되는 각 제품의 단위를 나타냅니다. 데이터 세트에는 811개 제품이 있습니다.

제품 코드별 최적 시계열과의 거리 계산

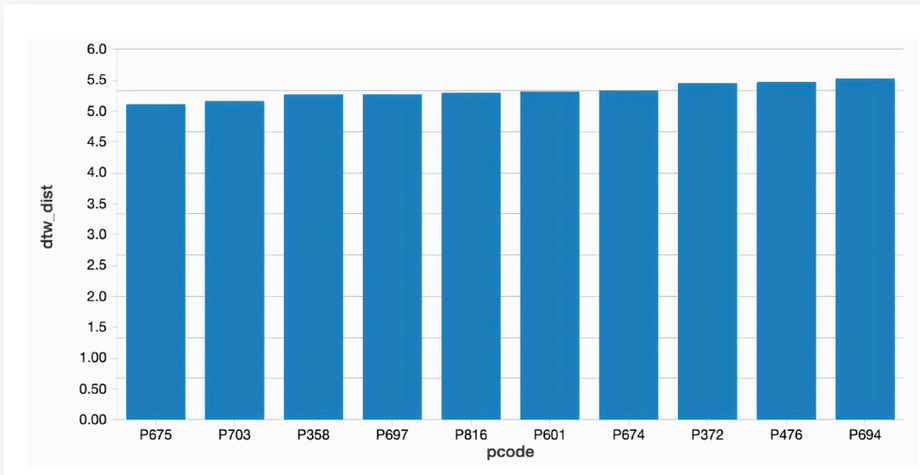
```
# Calculate distance via dynamic time warping between product code and
optimal time series
import numpy as np
import _ucrdtw

def get_keyed_values(s):
    return(s[0], s[1:])

def compute_distance(row):
    return(row[0], _ucrdtw.ucrdtw(list(row[1][0:52]), list(optimal_pattern),
0.05, True)[1])

ts_values = pd.DataFrame(np.apply_along_axis(get_keyed_values, 1, sales_pdf.
values))
distances = pd.DataFrame(np.apply_along_axis(compute_distance, 1, ts_values.
values))
distances.columns = ['pcode', 'dtw_dist']
```

계산된 동적 시간 왜곡 '거리' 열을 사용하면 DTW 거리의 분포를 히스토그램으로 확인할 수 있습니다.

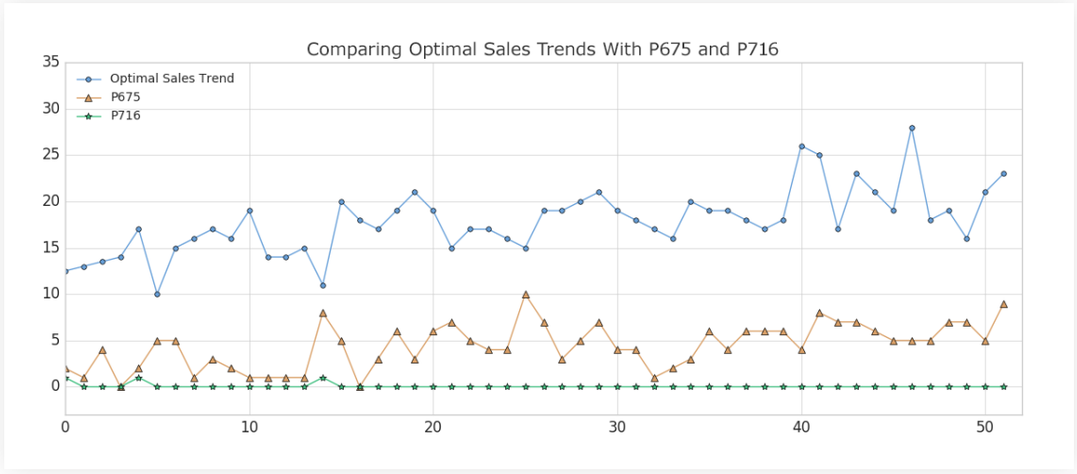


여기에서 최적 매출 동향과 가장 가까운 제품 코드를 찾을 수 있습니다(즉, 산출된 DTW 거리가 가장 작은 제품). 우리는 Databricks를 사용하고 있기 때문에 SQL 쿼리로 간편하게 선택할 수 있습니다. DTW 거리가 가장 가까운 제품을 표시해보겠습니다.

```
%sql
-- Top 10 product codes closest to the optimal sales trend
select pcode, cast(dtw_dist as float) as dtw_dist from distances order by cast(dtw_dist as float) limit 10
```



이 쿼리를 실행한 뒤, 최적 매출 동향에서 가장 멀리 떨어진 제품 코드에 대한 해당 쿼리를 실행해서 매출 동향으로부터 거리가 가장 가까운 제품과 가장 먼 제품 두 개를 찾았습니다. 그래프를 통해 두 제품에 어떤 차이가 있는지 살펴보겠습니다.

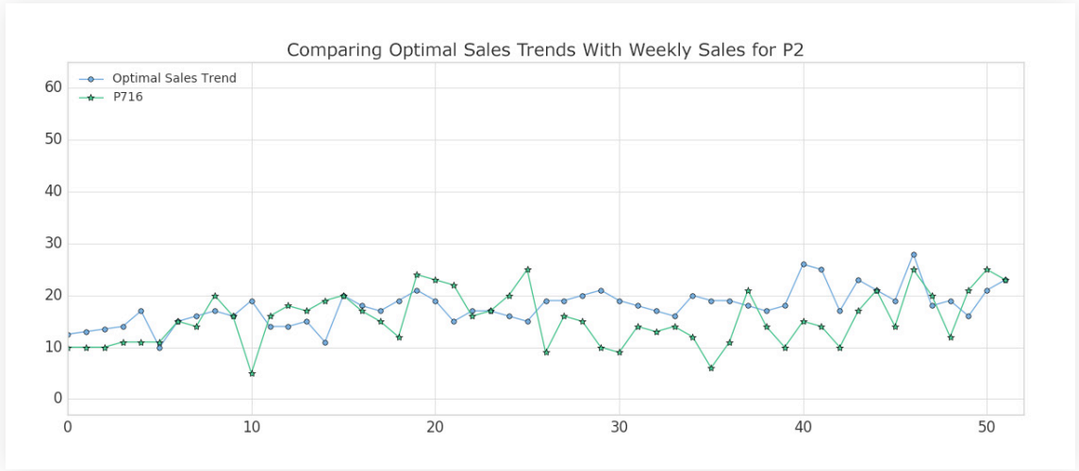


여러분도 볼 수 있듯이, 제품 #675(주황색 삼각형)는 최적 매출 동향과 가장 일치하지만 절대 주간 매출은 우리가 원하는 수준보다 낮습니다(이는 나중에 해결하겠습니다). 이런 결과를 이해할 수 있는 이유는 DTW 거리가 가장 가까운 제품은 비교 대상 지표를 다소 반영하여 고점과 저점이 있을 것으로 예상했기 때문입니다. (물론, 제품의 정확한 시간 색인은 동적 시간 왜곡으로 인해 매주 다릅니다). 반면, 제품 #716(녹색 별)은 가장 일치하지 않는 제품이라 거의 분산이 없습니다.

최적 제품 찾기: DTW 거리가 가깝고, 절대 매출이 유사한 제품

공장의 예상 생산량(“최적 매출 동향”)과 가장 가까운 제품 목록을 만들었으므로 DTW 거리가 가까우면서도 절대 매출이 유사한 제품만 필터링해보겠습니다. 가능성이 있는 후보로는 제품 #202가 있습니다. DTW 거리가 모집단 중간값은 7.89인데 비해 6.86이고, 최적 매출 동향을 매우 근사하게 따라갑니다.

```
# Review P202 weekly sales
y_p202 = sales_pdf[sales_pdf['Product_Code'] == 'P202'].values[0][1:53]
```



MLflow를 사용하여 아티팩트와 함께 가장 적절한 제품과 가장 부적절한 제품 추적

MLflow는 실험, 재현, 배포를 포함한 머신 러닝 수명 주기를 관리하기 위한 오픈 소스 플랫폼입니다. **Databricks 노트북은 완전 통합 MLflow 환경을 제공하므로, 실험을 생성하여 매개변수와 지표를 기록하고 결과를 저장할 수 있습니다.** 이 유용한 문서에서 MLFlow를 시작하는 방법에 대한 자세한 정보를 확인할 수 있습니다.

MLflow는 각 실험의 모든 입력값과 출력값을 체계적이고 재현할 수 있는 방식으로 기록하는 기능을 중심으로 설계됩니다. “런”이라고 하는 데이터를 통과시키는 과정을 실행할 때마다 실험값을 기록할 수 있습니다.

- **매개변수**: 모델에 대한 입력값
- **지표**: 모델의 출력값, 모델의 성공 측정
- **아티팩트**: 모델에서 생성한 모델 — 예: PNG 그래프, 또는 CSV 데이터 결과값
- **모델**: 모델 자체(나중에 다시 로드에서 예측하는 데 사용 가능)

우리 사례에서는 “신장 계수(stretch factor)”를 변경하면서 데이터에 동적 시간 왜곡 알고리즘을 여러 번 실행할 수 있습니다. 신장 계수란 시계열 데이터에 적용할 수 있는 최대 왜곡 횟수를 나타냅니다. MLflow 실험을 초기화하고 `mlflow.log_param()`, `mlflow.log_metric()`, `mlflow.log_artifact()`, `mlflow.log_model()` 를 사용하여 간편하게 로깅하기 위해 다음을 사용하여 메인 함수를 래핑했습니다.

```
iwith mlflow.start_run() as run:
    ...
```

요약된 코드는 오른쪽에 나와 있습니다.

```
import mlflow

def run_DTW(ts_stretch_factor):
    # calculate DTW distance and Z-score for each product
    with mlflow.start_run() as run:

        # Log Model using Custom Flavor
        dtw_model = {‘stretch_factor’ : float(ts_stretch_factor), ‘pattern’ : optimal_
pattern}
        mlflow_custom_flavor.log_model(dtw_model, artifact_path=”model”)

        # Log our stretch factor parameter to MLflow
        mlflow.log_param(“stretch_factor”, ts_stretch_factor)

        # Log the median DTW distance for this run
        mlflow.log_metric(“Median Distance”, distance_median)

        # Log artifacts - CSV file and PNG plot - to MLflow
        mlflow.log_artifact(‘zscore_outliers_’ + str(ts_stretch_factor) + ‘.csv’)
        mlflow.log_artifact(‘DTW_dist_histogram.png’)

    return run.info

stretch_factors_to_test = [0.0, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5]
for n in stretch_factors_to_test:
    run_DTW(n)
```

데이터에 ‘런’을 한 번 실행할 때마다 사용한 “신장 계수”의 로그와 DTW 거리 지표의 Z 점수를 기준으로 이상치로 분류한 제품의 로그를 생성했습니다. 또한, DTW 거리 히스토그램의 아티팩트(파일)도 저장했습니다. **이 실험적 런은 Databricks에 로컬로 저장되고 나중에 실험 결과를 다시 보고 싶을 때 액세스할 수 있습니다.**

MLflow에서 각 실험의 로그를 저장했으므로 결과를 살펴볼 수 있습니다. Databricks 노트북에서 오른쪽 상단 모서리에 있는 "런" 아이콘을 선택하여 각 런의 결과를 확인하고 비교합니다.

당연하게도 "신장 계수"를 높일수록 거리 계수가 감소합니다. 직관적으로 생각해도 타당합니다. 시간 색인을 앞이나 뒤로 왜곡하도록 알고리즘에 유연성을 부여할수록 데이터에 더욱 가까운 결과를 찾아냅니다. 기본적으로는 분산 대신 편향을 감수한 것입니다.

MLflow를 사용한 모델 로깅

MLflow는 실험 매개변수, 지표, 아티팩트(예: 그래프, CSV 파일)를 기록할 수 있을 뿐만 아니라, 머신 러닝 모델도 기록할 수 있습니다. MLflow 모델은 일관적 API를 구성하는 구조의 폴더로, 다른 MLflow 도구 및 기능과의 호환성을 보장합니다. 이런 상호운용성은 매우 강력해서 모델을 다양한 프로덕션 환경에 빠르게 배포할 수 있습니다.

MLflow는 여러 가지 일반적으로 사용하는 머신 러닝 라이브러리(예: scikit-learn, Spark MLlib, PyTorch, TensorFlow)에 대해 다양한 공통적 모델 "플레이버"가 미리 로드되어 있습니다. 이런 모델 플레이버를 사용하면 모델을 처음에 생성한 이후에 손쉽게 기록하고 다시 로드할 수 있습니다. 이 **블로그 게시물**에서 예시를 확인하세요. 예를 들어 scikit-learn을 사용해서 MLflow를 사용하여 모델을 로깅할 때는 실험 내에서 다음 코드를 실행하기만 하면 됩니다.

```
mlflow.sklearn.log_model(model=sk_model, artifact_path="sk_model_path")
```

MLflow는 "Python 함수" 플레이버도 제공합니다. 타사 라이브러리(예: XGBoost 또는 spaCy)의 모델이나 심지어 간단한 Python 함수 자체도 MLflow 모델로 저장할 수 있습니다. Python 함수 플레이버를 사용하여 생성한 모든 Python 모델을 동일한 에코시스템에서 사용할 수 있고 추론 API를 통해 다른 MLflow 도구와 상호작용할 수 있습니다. 모든 사용 사례에 대해 계획하기는 불가능하지만, Python 함수 모델 플레이버는 최대한 보편적이고 유연하도록 설계되었습니다. 사용자 정의 처리와 로직 평가가 가능해서 ETL에 매우 유용합니다. 물론 더욱 "공식적인" 모델 플레이버가 온라인에 배포되고 있지만, 일반 Python 함수 플레이버도 여전히 "다양한 목적"으로 중요하게 사용됩니다. 모든 종류의 Python 코드와 MLflow의 로버스트 추적 툴킷을 연결해주는 역할을 합니다.

Python 함수 플레이버를 사용한 모델 로깅 과정은 간단합니다. **모델이나 함수를 모델로 저장할 때는 한 가지 요구 사항만 지키면 됩니다. pandas DataFrame을 입력값으로 받고 DataFrame 또는 NumPy 배열을 반환해야 합니다.** 이 요구 사항을 준수하고 나서, 함수를 MLflow 모델로 저장할 때는 PythonModel에서 상속하는 클래스를 정의하고, 사용자 정의 함수로 `.predict() 메서드`를 재정의하면 됩니다. 이 내용은 **여기**를 참조하세요.

런에서 로깅된 모델 로드

여러 신장 계수로 데이터를 실행해보았으므로 다음 단계에서는 결과를 검토하고 로깅된 지표를 기준으로 가장 성능이 좋은 모델을 찾아야 합니다. **MLflow를 사용하면 쉽게 로깅된 모델을 다시 로드하고 새로운 데이터에 대해 예측하는 데 사용할 수 있습니다. 이때 다음과 같은 명령을 사용합니다.**

1. 모델을 로드할 런의 링크를 클릭합니다.
2. '런 ID'를 복사합니다.
3. 모델이 저장된 폴더 이름을 기록합니다. 이 경우, "model"이라고 했습니다.
4. 모델 폴더 이름과 런 ID를 아래와 같이 입력합니다.

```
import custom_flavor as mlflow_custom_flavor

loaded_model = mlflow_custom_flavor.load_model(artifact_path='model', run_
id='e26961b25c4d4402a9a5a7a679fc8052')
```

모델이 원하는 대로 작동하는지 확인하기 위해 모델을 로드해서 `new_sales_units` 변수 내에서 생성한 새로운 제품 두 가지에 대한 DTW 거리를 측정합니다.

```
# use the model to evaluate new products found in 'new_sales_units'
output = loaded_model.predict(new_sales_units)
print(output)
```

다음 단계

보다시피 MLflow 모델이 손쉽게 처음 보는 새로운 값을 예측합니다. 추론 API를 따르기 때문에 어떤 서비스 플랫폼(예: Microsoft Azure ML 또는 Amazon SageMaker)에나 모델을 배포할 수 있으며, 로컬 REST API 엔드포인트로 배포하거나 사용자 정의 함수(UDF)를 생성해 Spark-SQL에서 간편하게 사용할 수 있습니다. 지금까지 동적 시간 왜곡을 사용하여 Databricks Unified Data Analytics Platform에서 매출 동향을 예측하는 방법을 보여드렸습니다. 어서 동적 시간 왜곡과 MLflow를 사용하여 매출 동향 예측 노트북을 Databricks Runtime for Machine Learning을 사용해서 실행해보세요.

3장:

Prophet 및 Apache Spark를 사용한 세분화된 대규모 시계열 예측™

글: Bilal Obeidat, Bryan Smith 및 Brenner Heintz

2020년 1월 27일

Databricks에서 이 시계열 예측 노트북 보기 →

시계열 예측 기술이 발전한 덕분에 리테일러는 더욱 신뢰할 수 있는 방법으로 수요를 예측할 수 있게 되었습니다. 지금은 기업에서 정밀하게 제품 재고를 조정할 수 있을 정도로 이런 예측을 시기적절하고 세분화하여 수행해야 한다는 문제가 남아 있습니다. 이 문제에 직면한 기업들은 Apache Spark 및 Facebook Prophet을 사용하면서 과거의 솔루션에 존재하던 확장성과 정확도의 한계를 극복하고 있습니다.

이 게시물에서는 시계열 예측의 중요성을 설명하고, 몇 가지 샘플 시계열 데이터를 시각화한 다음, 간단한 모델을 구축해 Facebook Prophet으로 보여드릴 것입니다. 모델 하나를 구축하는 방법을 알고 나면 Prophet과 마법과도 같은 Apache Spark™를 결합하여 한 번에 수백 개의 모델을 훈련해보겠습니다. 이를 통해 개별 제품-매장 조합에 대해 지금까지는 보기 어려웠던 수준으로 세분화된 정밀한 예측을 예측할 수 있습니다.

정확하고 시기적절한 예측이 그 어느 때보다 중요

시계열 분석의 속도와 정확도를 개선하여 제품과 서비스의 수요 예측을 개선하는 것은 리테일러의 성공에 매우 중요합니다. 지나치게 많은 제품을 매장에 배치할 경우, 선반과 창고 공간이 부족해지고 제품 유통 기한이 지날 수 있습니다. 리테일러들은 재고에 자금이 묶여서 제조사나 소비자 패턴 변화에서 발생하는 새로운 기회를 놓칠 수도 있습니다. 매장에 제품이 너무 적으면 고객들은 필요한 제품을 구매하지 못할 수 있습니다. 이런 예측 오류는 리테일러에 수익 손실을 바로 입힐 뿐만 아니라, 장기적으로도 소비자 불만이 커져 경쟁사로 소비자가 이탈할 수도 있습니다.

새로운 기대로 인해 더욱 정밀한 시계열 예측 방법과 모델이 필요

한동안 기업 리소스 계획(ERP) 시스템과 타사 솔루션은 간단한 시계열 모델을 기반으로 수요 예측 기능을 제공했습니다. 하지만 기술이 발전하고 이 부문에서의 압력이 커지면서 많은 리테일러가 기존의 선형 모델과 더욱 전통적인 알고리즘에서 벗어날 방법을 모색하고 있습니다.

PROPHET

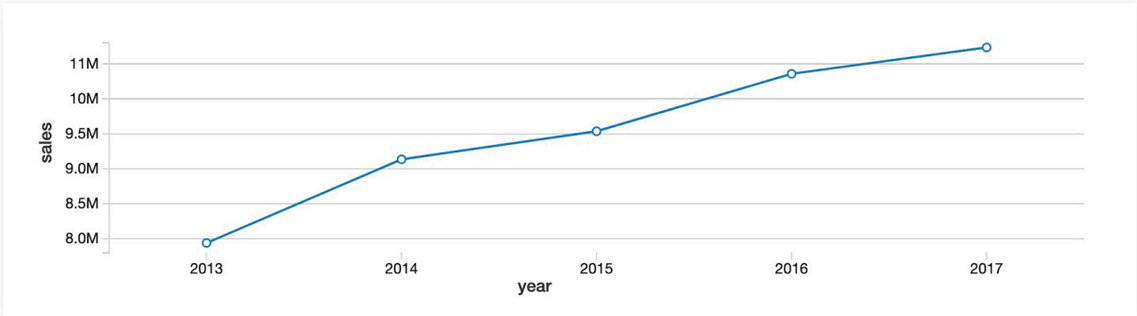
새로운 기능(예: Facebook Prophet에서 제공하는 기능)이 데이터 사이언스 계에 나타나고 있고, 기업들은 이런 머신 러닝 모델을 시계열 예측 요구 사항에 적용할 유연성을 얻고자 합니다.

리테일러 등이 기존 예측 솔루션에서 벗어나려면 조직 내부에서 수요 예측의 복잡성에 대한 전문성을 기르는 동시에, 시기적절하게 수십만, 심지어 수백만 개의 머신 러닝 모델을 생성하는 데 필요한 업무도 효율적으로 분담해야 합니다. 다행히 Spark를 사용하여 이 정도의 모델을 훈련할 수 있으므로 제품과 서비스에 대한 전반적 수요뿐만 아니라 각 위치에 있는 각 제품의 고유 수요도 예측할 수 있습니다.

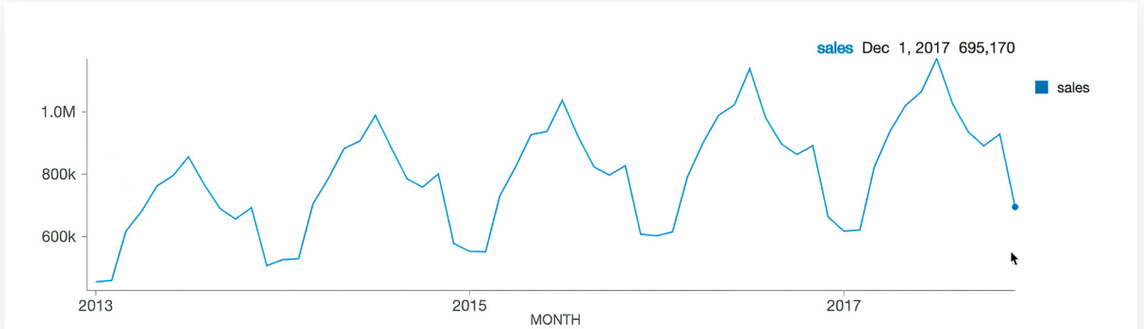
시계열 데이터에서 수요 계절성 시각화

Prophet을 사용하여 각 매장과 제품에 세분화된 수요 예측을 생성하는 방법을 보여드리기 위해 Kaggle에서 공개적으로 제공되는 **데이터 세트**를 사용할 것입니다. 이는 10개 매장에 있는 50개 품목에 대한 일일 매출 데이터 5년분으로 구성됩니다.

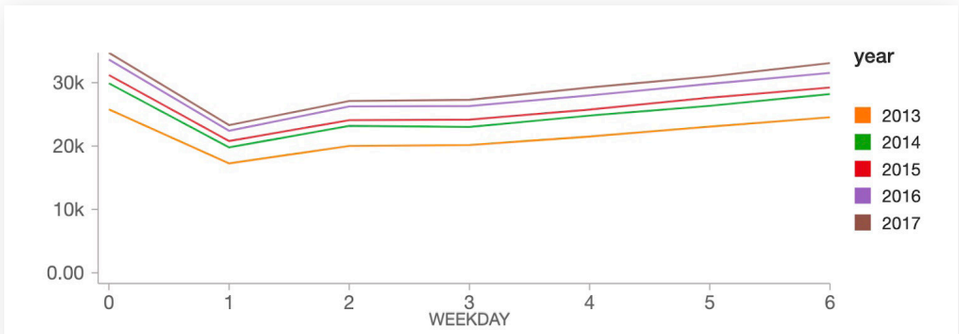
먼저 모든 제품과 매장의 전체 연간 매출 동향을 살펴보겠습니다. 여기에서 볼 수 있듯이, 전체 제품 매출은 매년 늘어나고 있고 일정 수준에서 수렴할 기미는 보이지 않습니다.



이제 같은 데이터를 월별로 살펴보면 매년 상승하는 흐름이 월 단위로는 일정하게 나타나지 않는 것을 알 수 있습니다. 그 대신 여름에는 계절적인 성수기 패턴이, 겨울에는 비수기 패턴이 명확하게 드러납니다. **Databricks Collaborative Notebooks**에 구축된 데이터 시각화 기능을 사용하면 그래프에 마우스를 가져가서 각 월의 데이터 값을 확인할 수 있습니다.



요일별로는 일요일(0일 차)에 매출이 가장 높았다가, 월요일(1일 차)에 급격히 하락한 다음, 주말이 올 때까지 서서히 회복됩니다.



Facebook Prophet에서 간단한 시계열 예측 모델을 사용해보겠습니다.

위의 그래프에서 볼 수 있듯이, 데이터에서는 매년 꾸준하게 매출이 상승하고 있고 연간, 주간 계절적 패턴을 동반합니다. Prophet은 이런 데이터의 중첩 패턴을 찾도록 설계되었습니다.

Facebook Prophet은 scikit-learn API를 따르기 때문에 sklearn을 사용해본 경험이 있는 사람이라면 누구나 쉽게 익힐 수 있습니다. 2열 pandas DataFrame을 입력값으로 전달해야 합니다. 첫 번째 열은 날짜이고 두 번째 열은 예측하기 위한 값(이 경우, 매출)입니다. 데이터 형식이 적절할 경우 쉽게 모델을 구축할 수 있습니다.

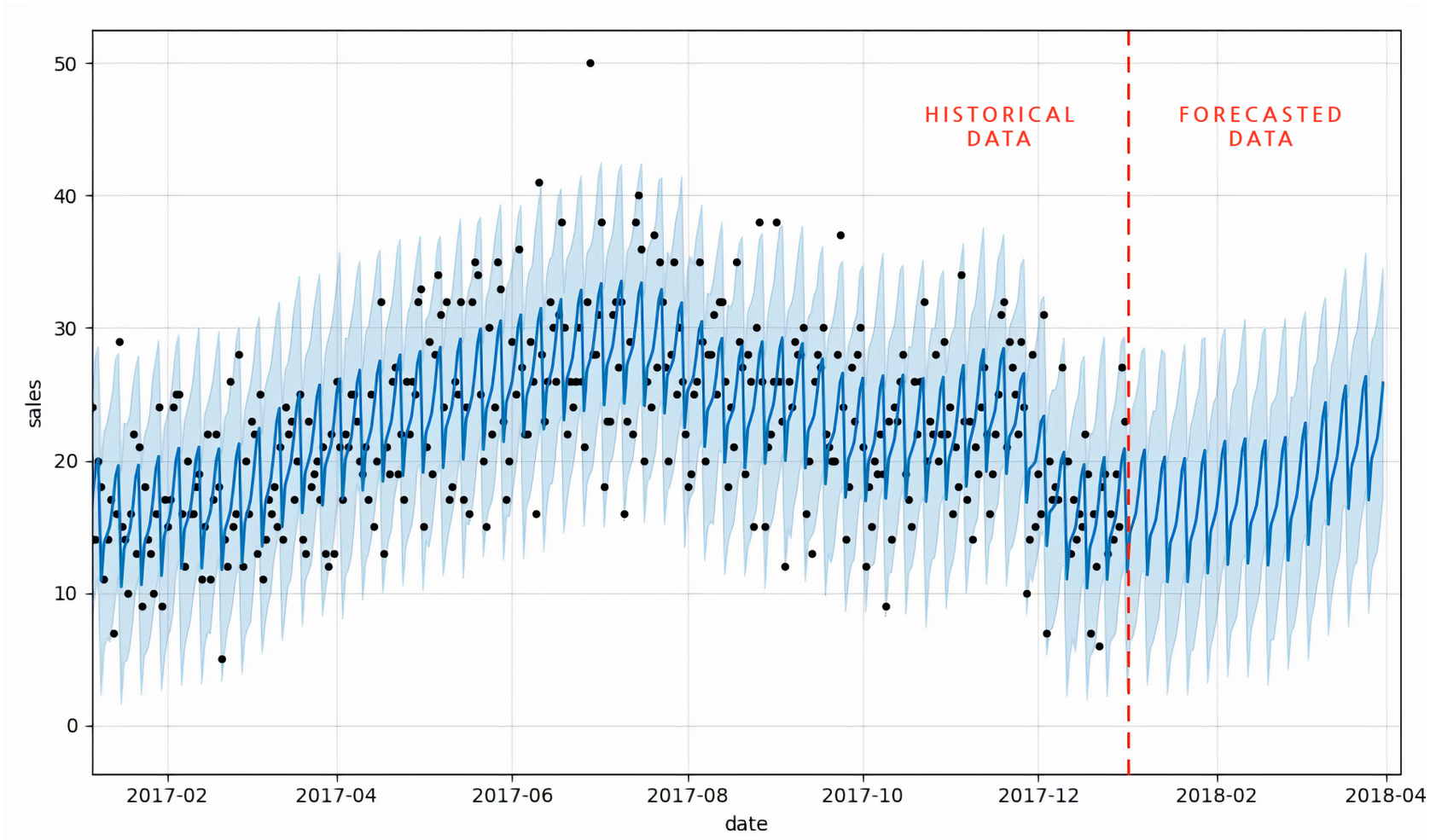
```
import pandas as pd
from fbprophet import Prophet

# instantiate the model and set parameters
model = Prophet(
    interval_width=0.95,
    growth='linear',
    daily_seasonality=False,
    weekly_seasonality=True,
    yearly_seasonality=True,
    seasonality_mode='multiplicative'
)

# fit the model to historical data
model.fit(history_pd)
```

이제 모델을 데이터에 피팅했으므로 90일 예측을 구축해보겠습니다. 아래의 코드에서 prophet의 `make_future_dataframe` 메서드를 사용하여 과거 날짜와 90일 이전을 포함하는 데이터 세트를 정의합니다.

```
future_pd = model.make_future_dataframe(
    periods=90,
    freq='d',
    include_history=True
)

# predict over the dataset
forecast_pd = model.predict(future_pd)
```

이것으로 끝입니다! Prophet의 내장 .plot 메서드를 사용하여 실제 및 예측 데이터가 어떻게 나타나고, 미래가 어떻게 예측되는지 시각화할 수 있습니다. 보다시피, 앞서 설명했던 주간 및 계절적 수요 패턴이 예측 결과에 반영되어 있습니다.

```
predict_fig = model.plot(forecast_pd, xlabel='date', ylabel='sales')
display(fig)
```



이 시각화 자료는 다소 복잡해 보입니다. Bartosz Mikulski가 **훌륭한 분석**을 제공하므로 한 번 확인해보시는 것이 좋습니다.
요약해서 말씀드리면, 검은색 점은 실제 데이터이고 남색 선은 예측을 나타냅니다. 하늘색 띠는 (95%) 불확도를 나타냅니다.

Prophet과 Spark를 사용하여 수백 개의 시계열 예측 모델 훈련

하나의 시계열 예측 모델을 구축하는 방법을 알아보았으므로, Apache Spark를 사용하여 모델을 늘려보겠습니다. 이번 목표는 전체 데이터 세트에 하나의 예측을 생성하는 것이 아니라, 각 제품-매장 조합에 대해 수백 개의 모델과 예측을 생성하는 것입니다. 하나씩 작업하려면 엄청난 시간이 소요될 것입니다.

이런 방식으로 모델을 구축하면 (가령) 슈퍼마켓 체인은 선더스키 매장과 클리블랜드 매장의 서로 다른 우유 주문량을 각 위치의 수요에 따라 정밀하게 예측할 수 있습니다.

Spark DataFrames을 사용하여 시계열 데이터 처리를 분산하는 방법

데이터 사이언티스트는 Apache Spark와 같은 분산된 데이터 처리 엔진을 사용하여 대량의 모델 훈련 문제를 해결하는 경우가 많습니다. Spark 클러스터를 사용하면 해당 클러스터에 있는 각 작업자 노드는 다른 작업자 노드와 동시에 모델의 하위 집합을 훈련할 수 있어서 전체 시계열 모델을 훈련하는 데 필요한 시간을 대폭 절약할 수 있습니다.

물론, 작업자 노드(컴퓨터)로 구성된 클러스터 하나에서 모델을 훈련하려면 더 많은 클라우드 인프라가 필요하고 여기에는 비용이 발생합니다. 하지만 온디맨드 클라우드 리소스를 손쉽게 사용할 수 있으므로 기업에서는 필요한 리소스를 재빨리 프로비저닝하여 모델을 훈련하고, 똑같이 빠른 속도로 리소스를 해제할 수도 있습니다. 따라서 물리적 자산에 장기적으로 투자하지 않더라도 엄청난 확장성을 달성할 수 있습니다.

Spark에서 분산된 데이터 처리를 지원하는 핵심 메커니즘은 DataFrame입니다. Spark DataFrame으로 데이터를 로드하면 클러스터의 작업자 전체에 데이터가 분산됩니다. 그러면 작업자가 동시에 데이터 하위 집합을 처리함으로써, 작업을 수행하는 데 필요한 전반적인 시간이 감소합니다.

물론, 각 작업자는 처리에 필요한 데이터 하위 집합에 액세스해야 합니다. 키 값(이 경우, 매장과 품목의 조합)에 대해 데이터를 그룹화하여 해당 키 값에 대한 모든 시계열 데이터를 특정 작업자 노드로 통합합니다.

```
store_item_history
    .groupBy('store', 'item')
    # . . .
```

> 얼마나 효율적으로 대량의 모델을 훈련할 수 있는지 보여드리기 위해 groupBy 코드를 공유합니다. 그러나 다음 섹션에서 UDF를 설정하고 데이터에 적용해야 실제로 어떨지 알 수 있습니다.

pandas 사용자 정의 함수 활용

매장과 품목을 기준으로 시계열 데이터를 적절히 그룹화하면 각 그룹에 하나의 모델을 훈련해야 합니다. 이를 위해서는 pandas 사용자 정의 함수(UDF)를 사용합니다. DataFrame에서 사용자 정의 함수를 각 데이터 그룹에 적용할 수 있습니다.

이 UDF는 각 그룹에 대해 모델을 훈련할 뿐만 아니라, 해당 모델에서 예측을 나타내는 결과 세트도 생성합니다. 이 함수는 다른 함수와 독립적으로 DataFrame에서 각 그룹에 대해 훈련하고 예측을 생성하지만, 각 그룹에서 반환된 결과를 하나의 DataFrame 결과로 편리하게 통합할 수 있습니다. 그러면 매장-품목 단위 예측을 생성하면서도 하나의 결과 데이터 세트로 분석 전문가와 관리자에게 제시할 수 있습니다.

축약된 Python 코드에서 볼 수 있듯이, UDF를 구축하는 방법은 비교적 간단합니다. UDF는 `pandas_udf` 메서드로 인스턴스화되고, 이 메서드는 반환할 데이터의 스키마와 수신할 데이터 타입을 식별합니다. 그다음에는 UDF의 작업을 실행할 함수를 정의하겠습니다.

함수를 정의하는 동안 모델을 인스턴스화해서 구성하고, 수신한 데이터에 피팅할 것입니다. 모델에서 예측하면 그 데이터는 함수의 결과값으로 반환됩니다.

```
@pandas_udf(result_schema, PandasUDFType.GROUPED_MAP)
def forecast_store_item(history_pd):

    # instantiate the model, configure the parameters
    model = Prophet(
        interval_width=0.95,
        growth='linear',
        daily_seasonality=False,
        weekly_seasonality=True,
        yearly_seasonality=True,
        seasonality_mode='multiplicative'
    )

    # fit the model
    model.fit(history_pd)

    # configure predictions
    future_pd = model.make_future_dataframe(
        periods=90,
        freq='d',
        include_history=True
    )

    # make predictions
    results_pd = model.predict(future_pd)

    # . . .

    # return predictions
    return results_pd
```

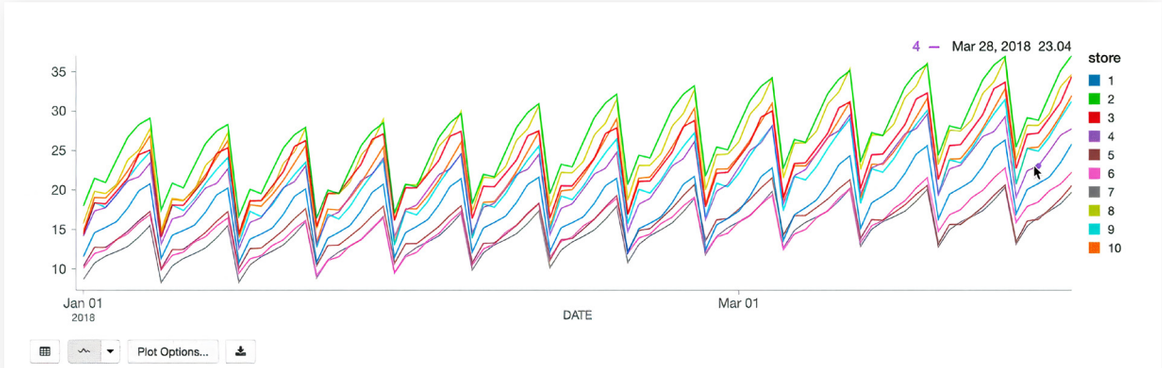
이제 모든 결과를 합치기 위해 앞서 설명한 `groupBy` 명령을 사용하여 데이터 세트가 각 매장과 품목 조합을 나타내는 그룹으로 적절하게 분할되었는지 확인합니다. 그런 다음에는 UDF를 DataFrame에 `적용`하여 UDF에서 모델을 피팅하고 각 데이터 그룹에 대해 예측하도록 합니다.

함수를 각 그룹에 적용하여 반환된 데이터 세트는 예측을 생성한 날짜를 반영하여 업데이트됩니다. 최종적으로 기능을 프로덕션으로 배포했을 때 각 모델 런에서 생성된 데이터를 추적하는 데 도움이 됩니다.

```
from pyspark.sql.functions import current_date

results = (
store_item_history
.groupBy('store', 'item')
.apply(forecast_store_item)
.withColumn('training_date', current_date())
)
```

다음 단계

이제 각 매장-품목 조합에 대해 시계열 예측 모델을 생성했습니다. 분석 전문가는 SQL 쿼리를 사용하여 각 제품에 대한 맞춤형 예측을 확인할 수 있습니다. 아래의 그래프는 10개 매장에서 제품 #1에 대한 예상 수요를 나타냅니다. 보다시피, 수요 예측은 매장마다 다르지만 일반적 패턴은 모든 매장에서 일치하는 것은 예상한 그대로입니다.



새로운 매출 데이터가 도착하면 새로운 예측을 효율적으로 생성하고 기존 표 구조에 첨부할 수 있습니다. 분석 전문가는 상황에 맞춰 기업의 기대를 업데이트할 수 있습니다.

자세한 내용은 **Starbucks에서 Facebook Prophet과 Azure Databricks를 사용하여 대규모 수요를 예측하는 방법** 온디맨드 웨비나를 참조하세요.

4장:

순환신경망을 사용한 다변량 시계열 예측

Keras의 장단기 메모리(LSTM) 구현을 사용한 시계열 예측

글: Vedant Jain

2019년 9월 10일

Databricks에서 노트북 보기 →

시계열 예측은 머신 러닝에서 중요한 분야입니다. 시계열 데이터의 성격으로 인해 정확한 모델을 구축하기 어려울 수 있습니다. 최근 들어 머신 러닝의 신경망 분야가 발전하면서 기존 시계열 예측 방식의 범위에서 벗어나거나 해결하기 어려웠던 다양한 문제를 해결할 수 있게 되었습니다. 이 게시물에서는 Keras의 **장단기 메모리(LSTM)**를 시계열 예측에 사용하고 MLflow를 모델 런 추적에 사용하는 방법을 알아보겠습니다.

LSTM이란 무엇일까요?

LSTM은 일종의 순환신경망(RNN)으로, 네트워크가 이전의 여러 시간 단계에서 지정된 시점에 장기 종속성을 유지할 수 있습니다. RNN은 뉴런에 대한 간단한 피드백 방식으로 이런 효과를 내도록 설계되었습니다. 여기서 데이터의 결과 순열이 입력값 중 하나의 역할을 합니다. 그러나 장기 종속성은 **기울기값이 사라지는 문제**로 인해 네트워크를 훈련할 수 없게 합니다. LSTM은 바로 이 문제를 해결하도록 설계되었습니다.

때로는 정확한 시계열을 예측하려면 과거 데이터와 최신 데이터의 약간씩 조합해야 할 경우도 있습니다. 무엇에 주의를 기울여야 할지도 효율적으로 학습하고, 오랜 기간의 과거 데이터가 있을 수 있다는 사실도 받아들여야 합니다. LSTM은 간단한 DNN 아키텍처와 지능적 메커니즘을 결합하여 장기에 걸쳐 과거의 어떤 부분을 “기억”하고 어떤 부분을 “잊을지” 학습합니다. LSTM은 긴 순열에서 데이터의 패턴을 학습하는 기능이 있어서 시계열 예측에 유리합니다.

LSTM 아키텍처의 이론적 기반은 **여기**(4장)를 참조하세요.

적절한 문제와 적절한 데이터 세트 선택

시계열을 활용하는 방법은 수없이 많습니다. 미래의 펀드 가격을 기반으로 포트폴리오를 구성하는 것에서부터 전기 공급망 등에 대한 수요를 예측하는 것까지 다양합니다. LSTM의 가치를 알 수 있으려면 적절한 문제, 특히 적절한 데이터 세트가 있어야 합니다. 미리 집의 습도와 온도에 대해 파악해서 스마트 센서가 A/C를 미리 켜게 하거나, 전기 소비량을 파악해서 미리 비용을 절감하고 싶은 경우를 생각해보세요. 과거의 센서 및 온도 데이터로도 이런 관계를 충분히 파악할 수 있고, LSTM이 여기에 도움이 될 수 있습니다. 최근 센서 값에 따라 달라질 뿐만 아니라, 특히나 오래된 값, 그 전날의 동일한 시간대에 얻은 센서 값에 따라 달라질 수 있기 때문입니다. 그래서 저전력 건물에서 장치의 에너지 사용량에 대한 실험적 데이터를 사용해보겠습니다.

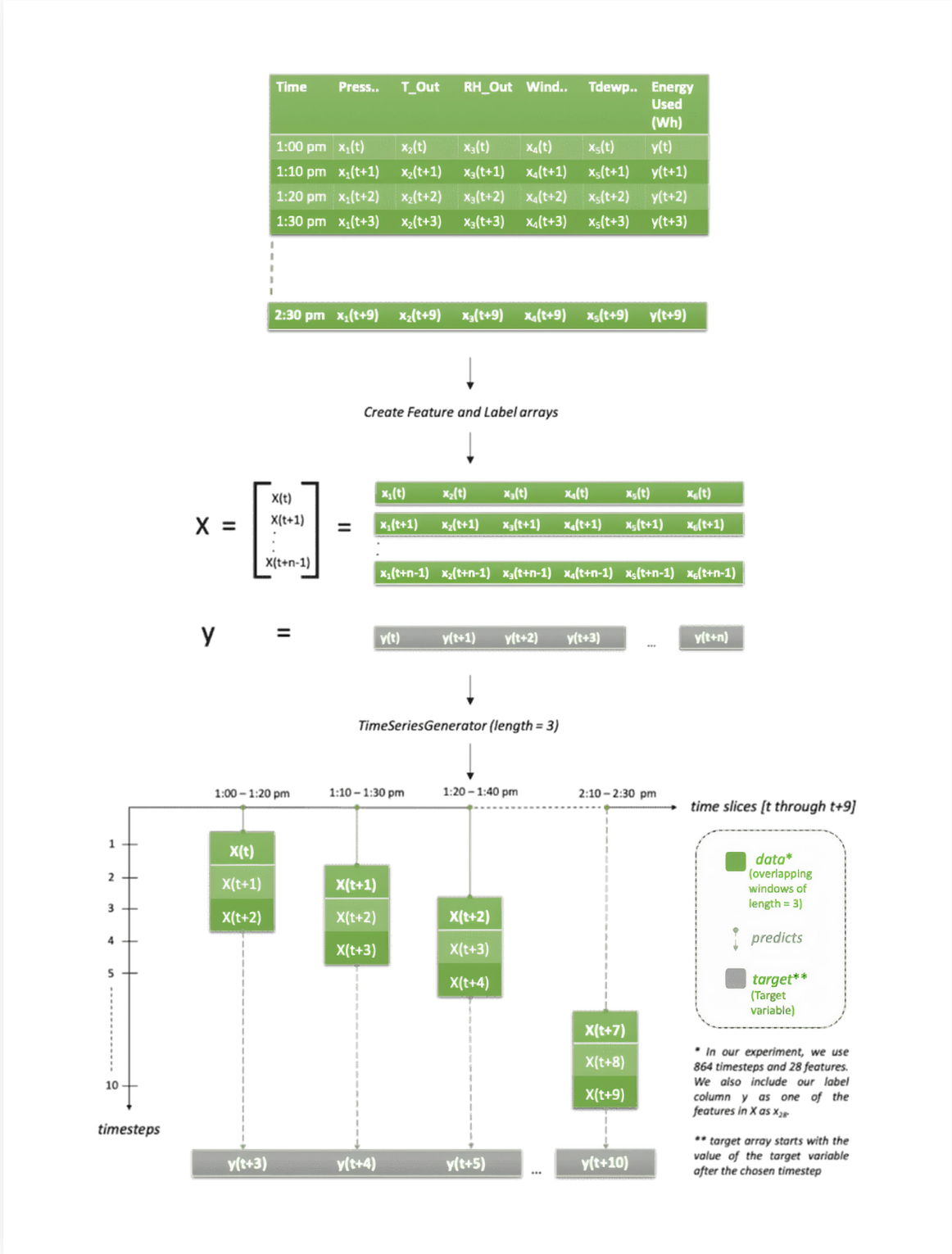
실험

이 실험에 선택한 **데이터 세트**는 기계의 에너지 사용량에 대한 회귀 모델을 구축하는 데 이상적입니다. 내부 온도와 습도는 ZigBee 무선 센서 네트워크로 모니터링했습니다. 약 4.5 개월 동안 10분간의 간격을 두고 측정했습니다. 에너지 데이터는 m-bus 에너지 계측기로 기록되었습니다. 가장 가까운 공항 기상청(벨기에, 치브르 공항)의 날씨를 Reliable Prognosis(rp5.ru) 공개 데이터 세트에서 다운로드하였고, 날짜와 시간 열을 사용하여 실험적 데이터 세트와 병합했습니다. 이 데이터 세트는 **UCI Machine Learning 리포지토리**에서 다운로드할 수 있습니다.

이를 사용하여 다음날 가전에서 소모할 에너지를 예측할 것입니다.

데이터 모델링

신경망을 훈련하기 전에, 네트워크가 과거 값의 순열에서 학습할 수 있도록 데이터를 모델링해야 합니다. 특히, LSTM은 입력 속성의 개수 만큼 타임스텝별로 테스트 샘플 크기의 특정 3D 텐서 형식으로 만든 입력 데이터가 필요합니다. 지도 학습 방식에서 LSTM이 학습을 시작하려면 feature와 레이블이 모두 필요합니다. 시계열 예측의 맥락에서 과거 값은 feature로, 미래의 값은 레이블로 제공하는 것이 중요합니다. 그래야 LSTM이 미래를 예측하는 방법을 배울 수 있습니다. 따라서 시계열 데이터를 'X'라는 feature의 2D 배열로 바꾸었고, 여기서 입력 데이터는 원하는 시간 단계에서 지연된 값을 일괄 중첩한 것으로 구성됩니다. 각 입력 feature의 배치에 대해 예측하려는 레이블이나 미래의 값만으로 구성된 'y'라는 1D 배열을 생성합니다. 또한, 입력 데이터에는 'y'의 지연된 값을 포함하여 네트워크가 레이블의 과거 값으로부터 학습할 수 있도록 했습니다. 자세한 설명은 다음의 이미지를 참조하세요.



우리 데이터 세트에는 10분 샘플이 있습니다. 위의 이미지에서는 길이 = 3을 선택했습니다. 이는 모든 순열에서 30분의 데이터가 있다는 것을 의미합니다(10분 간격). 이 논리를 따르면, feature 'X'는 텐서 값 [X(t), X(t+1), X(t+2)], [X(t+2), X(t+3), X(t+4)], [X(t+3), X(t+4), X(t+5)]…등이어야 합니다. 목표 변수인 'y'는 [y(t+3), y(t+4), y(t+5)…y(t+10)]가 되어야 합니다. 시간 단계 또는 길이의 수가 3이 되어야 하므로 y(t), y(t+1), y(t+2) 값을 무시하겠습니다. 이 그래프에서는 각 입력 행에 대해 미래의 값 1개만 예측하는 것을 알 수 있습니다(즉, y(t+n+1)). 그러나 더욱 현실적인 시나리오에서는 더 먼 미래까지 예측할 수 있으며(y(t+n+L)), 이는 아래의 예시에서 확인할 수 있습니다.

Keras API는 시간 데이터를 중첩한 배치를 생성하는 TimeSeriesGenerator라는 클래스가 내장되어 있습니다. 이 클래스는 동일한 간격으로 수집된 데이터 포인트의 순열과 시계열 매개변수(예: 스트라이드, 과거의 기간)를 함께 받아서 훈련/검증을 위한 배치를 생성합니다.

그러므로 우리 사용 사례의 경우, 6일분의 과거 데이터에서 예측하는 방법을 학습하고 가령 1일 후 미래의 값을 예측해보겠습니다. 이 경우, 길이는 864이며, 이는 6일 내 10분 시간 단계의 개수(24x6x6)를 나타냅니다. 마찬가지로 습도, 온도, 압력 등의 과거 값에서도 학습하기를 원합니다. 즉, 모든 레이블에 대해 feature당 864개 값이 있습니다. 우리 데이터 세트에는 총 28개의 feature가 있습니다. 시간 순열을 생성할 때 생성 도구는 매번 6일분의 데이터로 구성되는 배치를 반환하도록 구성됩니다. 더욱 현실적인 시나리오로 만들기 위해 (10분 간격 대신) 1일 후의 사용량을 예측하기로 했고, 목표 값이 미래의 144개 시간 단계 값 세트(24x6x1)가 되도록 데이터 세트를 테스트하고 훈련할 준비를 했습니다. 자세한 내용은 노트북 섹션 2: 데이터 세트 정규화 및 준비를 참조하세요.

입력 세트의 형태는 (samples, timesteps, input_dim) [https://keras.io/layers/recurrent/]가 되어야 합니다. 모든 배치에 대해 6일분의 데이터, 즉 864개 행이 있습니다. 배치 크기는 기울기값을 업데이트하기 전에 샘플 값을 결정합니다.

```
# Create overlapping windows of lagged values for training and testing datasets
timesteps = 864
train_generator = TimeseriesGenerator(trainX, trainY, length=timesteps, sampling_
rate=1, batch_size=timesteps)
test_generator = TimeseriesGenerator(testX, testY, length=timesteps, sampling_rate=1,
batch_size=timesteps)
```

모든 튜닝 매개변수의 목록은 **여기**를 참조하세요.

모델 훈련

LSTM은 신경망에서 장기 종속성 문제를 해결할 수 있는데, **시간 경과에 따른 역전달(BPTT)**이라는 개념을 사용합니다. BPTT에 대한 자세한 내용은 **여기**를 참조하세요.

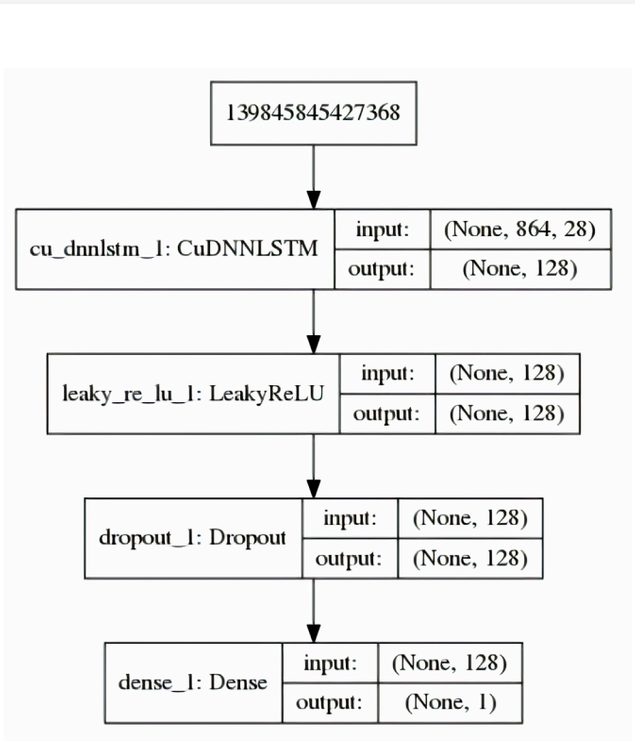
LSTM 네트워크를 훈련하기 전에 Keras에서 네트워크의 품질을 결정하는 몇 가지 중요한 매개변수를 이해해야 합니다.

1. **epoch:** 신경망에 데이터를 전달하는 횟수
2. **epoch당 단계:** 훈련 epoch가 완료된 것으로 간주하기 전까지 배치 반복 횟수
3. **활성화:** 사용할 **활성화 함수**를 설명하는 계층
4. **옵티마이저:** Keras는 내장된 **옵티마이저** 제공

```
units = 128
num_epoch = 5000
learning_rate = 0.00144

with mlflow.start_run(experiment_id=3133492, nested=True):

  model = Sequential()
  model.add(CuDNNLSTM(units, input_shape=(train_X.shape[1], train_X.shape[2])))
  model.add(LeakyReLU(alpha=0.5))
  model.add(Dropout(0.1))
  model.add(Dense(1))

adam = Adam(lr=learning_rate)
# Stop training when a monitored quantity has stopped improving.
callback = [EarlyStopping(monitor="loss", min_delta = 0.00001, patience = 50, mode =
'auto', restore_best_weights=True), tensorboard]

# Using regression loss function 'Mean Standard Error' and validation metric 'Mean
Absolute Error'
model.compile(loss="mse", optimizer=adam, metrics=['mae'])
    )
```

GPU의 속도와 성능을 활용하기 위해 CUDNN LSTM 구현을 사용합니다. 또한, 임의로 높은 epoch를 선택했습니다. 이는 데이터가 최적의 모델 핏을 찾아내기 위해 최대한 많은 반복(iteration)을 실행하기 위해서입니다. 단위 개수의 경우, feature가 28개이므로 32개로 시작합니다. 몇 번의 반복 후에 128개를 사용하면 적절한 결과를 얻을 수 있다는 것을 알았습니다.

epoch 수를 선택하려면 피팅이 부족해지지 않도록 높은 숫자를 선택하는 것이 좋습니다. 과도한 피팅이 발생하는 문제를 피하려면 Keras API에서 내장된 콜백을 사용할 수 있습니다. 특히, EarlyStopping을 사용하면 됩니다. EarlyStopping은 모니터링된 수치가 더 이상 개선되지 않으면 모델 훈련을 중단합니다. 우리 사례에서는 손실을 모니터링된 수치로 사용하고, 모델은 50 epoch당 1e-5의 감소가 없으면 훈련을 중단합니다. Keras는 내장 정규화기(regulizer)(가중치, 드롭아웃)가 있어서 매개변수가 매끄럽게 배포되도록 신경망에 불이익(penalize)을 줍니다. 그래서 신경망이 단위 간에 전달된 컨텍스트에 지나치게 의존하지 않도록 합니다. 아래의 이미지에서 신경망 계층을 참조하세요.



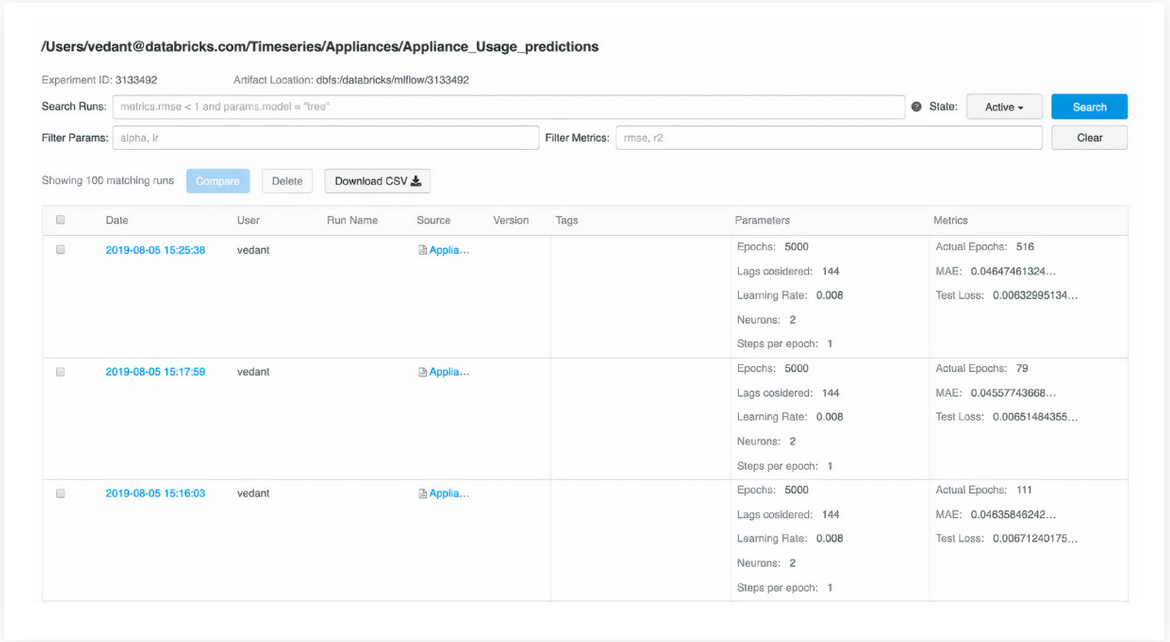
한 계층의 결과를 다른 계층으로 보내려면 활성화 함수가 필요합니다. 이 경우에는 LeakyRelu를 사용하는데, 이전 버전인 Rectifier Linear Unit(줄여서 Relu)보다 개선된 버전입니다.

Keras는 epoch에 대해 손실을 줄이기 위한 여러 가지 옵티마이저를 제공하고 epoch를 반복하면서 가중치를 업데이트합니다. 모든 옵티마이저 목록은 여기를 참조하세요. 우리는 Adam 확률적 기울기 강하 버전을 선택했습니다.

옵티마이저의 중요한 매개변수는 learning_rate로, 모델의 품질을 대규모로 확인할 수 있습니다. 학습률에 대한 자세한 내용은 여기를 참조하세요. 0.001(기본값), 0.01, 0.1 등의 다양한 값으로 실험을 하였고 0.00144가 훈련 속도와 최소 손실 면에서 최적의 모델 성능을 제공한다는 것을 알아냈습니다. 또한, LearningRateSchedular 콜백을 사용해서 학습률을 최적 값으로 조정할 수 있습니다. MLflow를 사용하여 여러 모델 런에서 결과를 추적, 비교할 수 있습니다.

MLflow를 사용한 모델 평가 및 로깅

보다시피, Keras LSTM 구현은 상당히 많은 하이퍼 매개변수를 받습니다. 최적의 모델 핏을 찾아내려면 다양한 하이퍼 매개변수(즉, 단위, epoch)로 실험해야 합니다. 또한, 과거의 모델 런을 비교하고 시간과 데이터 변화에 따른 모델 동작을 측정해야 합니다. MLflow는 위의 기능 외에도 다양한 기능을 제공하고 UI가 편리한 좋은 도구입니다. MLflow를 사용해서 얼마나 쉽게 Keras와 TensorFlow를 개발하고, MLflow를 로깅하고, 시간에 따라 실험을 추적할지 알 수 있습니다.



데이터 사이언티스트는 MLflow를 사용해서 다양한 모델 지표와 추가적 시각화, 아티팩트를 추적하고 프로덕션에서 어느 모델을 배포할지 결정하는 데 도움을 받을 수 있습니다. 이들은 두 개 이상의 모델 런을 비교하여 다양한 하이퍼 매개변수의 영향을 이해한 후 가장 최적의 모델을 결정합니다.

그러면 데이터 엔지니어는 선택한 모델, 훈련에 사용할 라이브러리 버전을 프로덕션의 새로운 데이터로 쉽게 가져올 수 있습니다. 최종 모델은 python_function 플레이버로 고정할 수 있습니다. 나중에는 Apache Spark UDF로 사용할 수도 있습니다. 이는 Spark 클러스터에 업로드하고 미래의 데이터를 평가하는 데 사용됩니다. MLflow에서 지원하는 모델 플레이버의 전체 목록은 여기를 참조하세요.

요약

- LSTM은 과거 값을 학습하여 미래를 예측하는 데 사용할 수 있습니다. LSTM은 기존의 방식과 달리 시계열에 대해 특정한 가정을 하지 않기 때문에 시계열 문제를 모델링하고 여러 입력값 중에서 비선형적 종속성을 학습하기가 더욱 쉽습니다.
- LSTM 네트워크에 입력하기 전에 이벤트의 순열을 생성할 때는 입력값보다 레이블을 지연시켜서 LSTM 네트워크가 과거 데이터로부터 학습하도록 하는 것이 중요합니다. Keras의 TimeSeriesGenerator 클래스를 사용하면 다양한 매개변수로 시계열 데이터 세트를 준비하여 변환한 다음, 시간이 지연된 데이터 세트를 신경망에 입력할 수 있습니다.
- LSTM은 튜닝할 수 있는 하이퍼 매개변수(예: epoch, 배치 크기)가 다양하며, 이는 예측의 품질을 확인하는 데 필요합니다. 학습률은 모델 가중치를 업데이트하는 방법과 모델의 학습 속도를 조절하는 중요한 하이퍼 매개변수입니다. 최적의 학습률 값을 찾아내서 최적의 모델 성능을 얻는 것이 중요합니다. LearingRateSchedular 콜백 매개변수를 사용하여 학습률을 최적 값으로 조정하는 것이 좋습니다.
- Keras는 해결하고자 하는 타입 문제에 사용할 수 있는 다양한 옵티마이저를 제공합니다. 일반적으로는 Adam이 적절합니다. MLflow UI를 사용하면 모델 런을 나란히 비교하고 최적의 모델을 선택할 수 있습니다.
- 시계열의 경우, 데이터에서 시간성을 유지하여 LSTM 신경망이 올바른 이벤트 순열에서 패턴을 학습할 수 있도록 하는 것이 중요합니다. 그러므로 테스트 및 검증 세트를 생성할 때와 모델을 피팅할 때 데이터를 섞지 말아야 합니다.
- 다른 모든 머신 러닝 기술과 마찬가지로, LSTMI도 잘못된 피팅은 어쩔 수 없습니다. 그래서 Keras에서는 EarlyStopping 콜백을 제공합니다. 약간의 직관과 적절한 콜백 매개변수가 있다면 하이퍼 매개변수를 튜닝하는 데 지나친 노력을 기울이지 않고도 적절한 모델 성능을 얻을 수 있습니다.
- RNN, 특히 LSTM은 대량의 데이터가 입력되었을 때 가장 잘 작동합니다. 데이터가 적을 때는 숨겨진 계층이 몇 개 있는 소규모 신경망으로 시작하는 것이 좋습니다. 데이터 규모가 작을 경우, epoch마다 데이터 배치를 키워서 제공하면 더 나은 결과가 나올 수 있습니다.

5장:

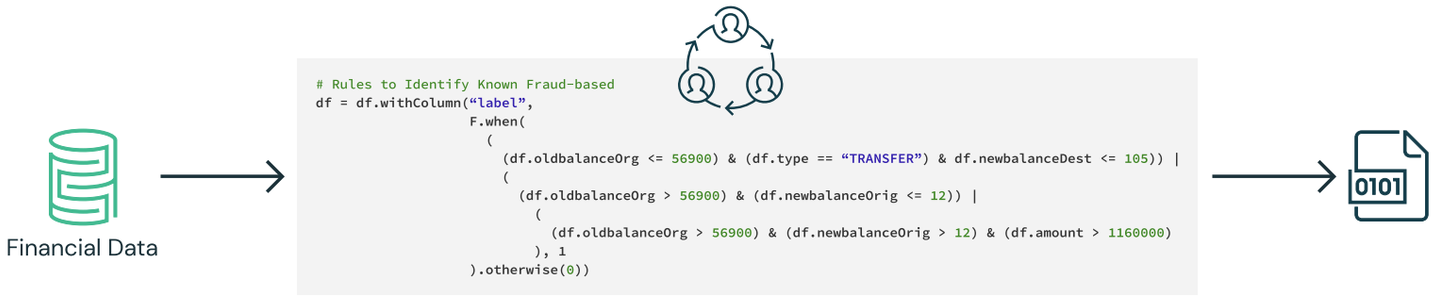
Databricks에서 결정 트리 및 MLflow로 대규모 금융 사기 탐지

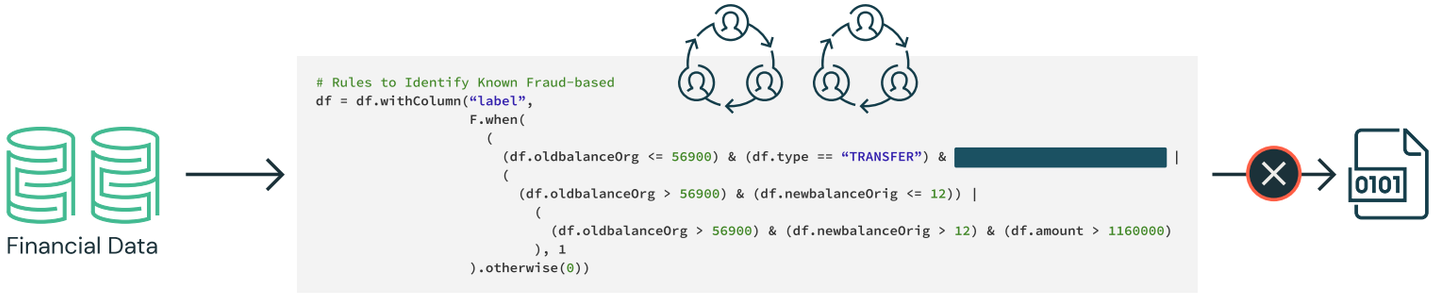
글: Elena Boiarskaia, Navin Albert 및 Denny Lee

2019년 5월 02일

Databricks에서 노트북 보기 →

어떤 사용 사례에서나 인공 지능을 사용하여 대규모 사기 패턴을 탐지하기란 어렵습니다. 대량의 과거 데이터를 검토해야 하고, 머신 러닝과 딥러닝 기술이 끊임없이 발전하면서 복잡성이 커지는 데다 사기 행위에 대한 매우 적은 실제 사례만으로 바늘이 어떻게 생겼는지도 모르면서 건초에서 바늘 찾듯이 찾아야 합니다. 금융 서비스 산업에서 보안에 대한 우려와 더불어 사기 행위를 찾아내는 방법을 설명해야 할 중요성이 커지면서 이 작업은 더욱 복잡해졌습니다.

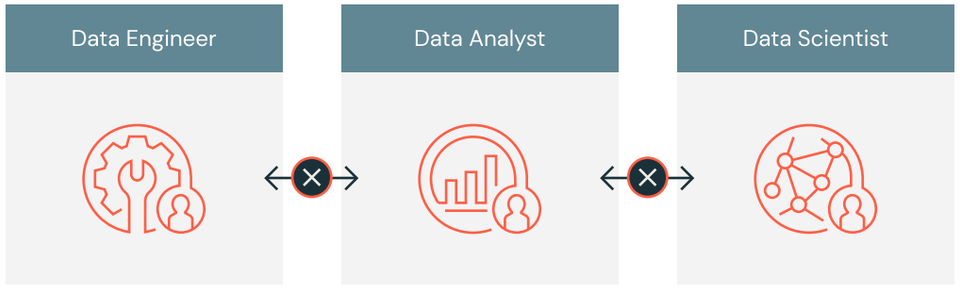




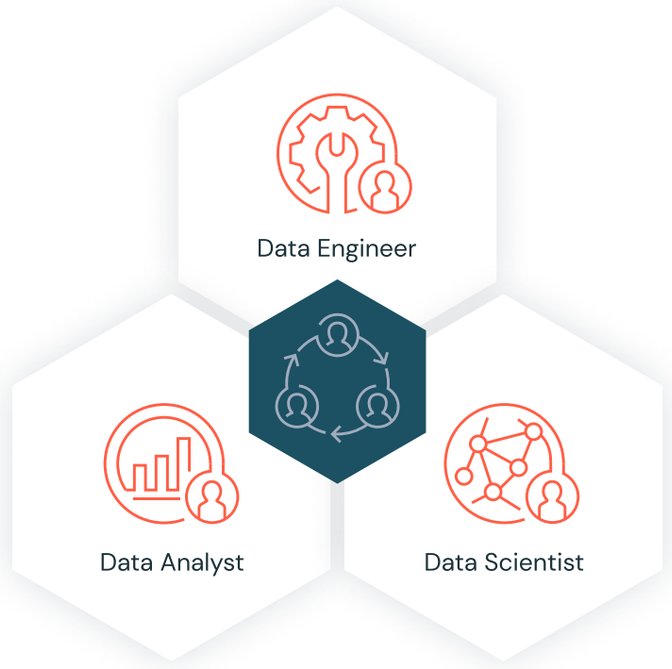
이런 탐지 패턴을 구축하기 위해 해당 분야 전문가팀이 사기범의 일반적인 행동을 바탕으로 규칙 세트를 만들었습니다. 워크플로에는 금융 사기 탐지 분야의 주제 전문가가 참여하여 특정 행동에 대한 요구 사항들을 작성할 수도 있습니다. 데이터 사이언티스트는 사용할 수 있는 데이터의 하위 샘플을 받아서 이러한 요구 사항과 일부 알려진 사기 사건을 활용하여 딥러닝 또는 머신 러닝 알고리즘 세트를 선택합니다. 데이터 엔지니어는 그 결과로 얻은 모델을 임계값을 적용한 규칙 세트로 변환하여 프로덕션에 패턴을 배포합니다. 이는 대부분 SQL을 사용하여 구현합니다.

databricks

이 방법을 사용하면 금융 기관에서 명확한 특징들을 파악하고, 이를 활용해 일반 데이터 보호 규정(GDPR)에 따라 사기성 거래를 찾아낼 수 있습니다. 그러나 그 과정에는 여러 가지 어려움이 따르기 마련입니다. 하드코딩된 규칙 세트로 사기 탐지 시스템을 구현하면 매우 불안정합니다. 사기 패턴이 바뀌었을 때 업데이트에 매우 오랜 시간이 걸리게 됩니다. 그러면 현재 시장에서 발생하는 사기성 행위의 변화를 따라잡고, 이에 적응하기 어렵게 됩니다.



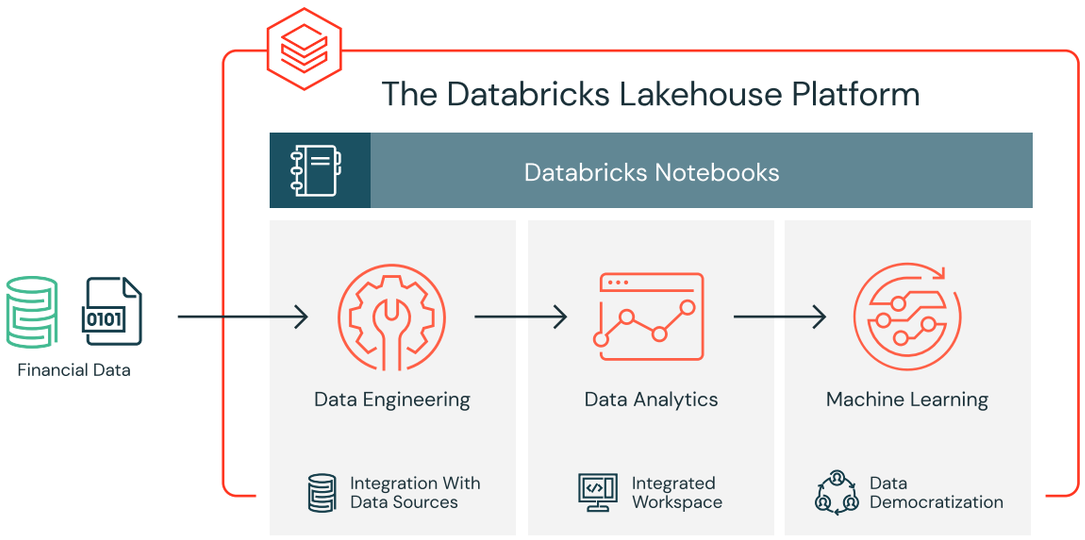
게다가 앞서 설명한 워크플로의 시스템은 대개 사일로화되어 있고 분야 전문가, 데이터 사이언티스트, 데이터 엔지니어가 모두 제각각 일합니다. 데이터 엔지니어는 엄청난 양의 데이터를 관리하고 분야 전문가와 데이터 사이언티스트의 작업을 프로덕션 코드로 변환해야 합니다. 공통적인 플랫폼이 없기 때문에 분야 전문가와 데이터 사이언티스트는 컴퓨터 하나에 들어갈 정도로 추출된 데이터를 사용해서 분석해야 합니다. 따라서 소통에 난맥이 생기고, 궁극적으로는 협업이 어려워집니다.



이 블로그에서는 Databricks 플랫폼에서 이런 여러 가지 규칙 기반 탐지 사용 사례를 머신 러닝 사용 사례로 변환하고, 사기 탐지 분야의 주요 담당자(분야 전문가, 데이터 사이언티스트, 데이터 엔지니어)를 통합하는 방법을 살펴봅니다. 머신 러닝 사기 탐지 데이터 파이프라인을 만들고, 프레임워크를 실시간 활용하여 데이터를 시각화함으로써 대규모 데이터 세트에서 모듈식 기능을 구축하는 방법을 배울 것입니다. 또한, 결정 트리와 Apache Spark™ MLlib를 사용한 사기 탐지 방법을 알아보겠습니다. 그런 다음, MLflow로 모델을 반복 개선하고 정확도를 높일 것입니다.

머신 러닝으로 문제 해결

금융 산업에서는 머신 러닝 모델과 관련하여 다소 꺼리는 분위기가 있습니다. 사기 사례를 발견하더라도 근거를 제공하지 못하는 "블랙박스"라고 생각하기 때문입니다. 데이터 사이언스를 활용하는 것은 GDPR 요구 사항과 금융 규제로 인해 불가능에 가까운 영역으로 보입니다. 그러나 여러 건의 성공적인 사용 사례에서 볼 수 있듯이, 머신 러닝을 대규모 사기 탐지에 적용하면 앞서 설명한 여러 가지 문제를 해결할 수 있습니다.



금융 사기를 탐지하기 위한 지도 머신 러닝 모델을 훈련하는 것은 실제 사기 행각이 발각된 사례가 적어서 매우 어렵습니다. 그러나 특정 사기 타입을 식별하는 규칙 세트가 알려져 있기 때문에 합성 레이블 세트와 최초 feature set을 만드는 데는 도움이 됩니다. 이 분야의 전문가가 개발한 탐지 패턴의 결과값은 적절한 승인 절차를 거쳐 프로덕션에 배포되었을 것입니다. 예상되는 사기 행위 플래그를 생성하므로 이를 시작점으로 삼아 머신 러닝 모델을 훈련할 수 있습니다.

이 방법은 세 가지 우려 사항을 동시에 완화합니다.

1. 바로 훈련 레이블이 부족하고,
2. 사용할 feature를 결정해야 하고,
3. 모델에 대한 적절한 벤치마크가 있어야 한다는 문제를 해결해줍니다.

규칙 기반 사기 행위 플래그를 인식하도록 머신 러닝 모델을 훈련하면 오차 행렬을 통해 예상되는 결과값과 직접 비교할 수 있습니다. 규칙이 규칙 기반 탐지 패턴과 매우 근사하게 일치하는 경우, 회의적인 태도를 보이는 사람들이 머신 러닝 기반 사기 예방의 효과를 신뢰하는 데 도움이 됩니다. 이 모델의 결과값은 매우 해석하기 쉽고, 원래 탐지 패턴과 비교하면 예상 거짓 양성과 거짓 음성에 대한 기본적인 논의가 가능할 수 있습니다.

게다가 머신 러닝 모델을 해석하기 어렵다는 우려는 결정 트리 모델을 최초 머신 러닝 모델로 사용하면 더욱 완화할 수 있습니다. 이 모델은 규칙 세트에 대해 훈련되므로 결정 트리가 다른 어떤 머신 러닝 모델보다도 더 나은 성과를 보일 것입니다. 물론, 모델이 가장 투명하다는 장점도 있습니다. 기본적으로 인간이 개입하거나 규칙이나 임계값을 하드코딩하지 않아도 사기에 대한 의사결정 과정을 보여줄 수 있습니다. 나중에 모델을 반복 개선하는 동안 다른 알고리즘까지 활용하면 정확도를 최대로 높일 수도 있습니다. 궁극적으로 모델의 투명성은 알고리즘에 입력된 feature를 이해해야 가능합니다. 해석할 수 있는 feature가 있으면 해석 가능하고 논리적인 모델 결과를 얻게 됩니다.

머신 러닝 기술의 가장 큰 장점은 처음에 모델링에 노력을 기울이면 나중에 모듈식 반복 개선을 통해 레이블 세트, feature, 모델 타입을 매우 쉽고 매끄럽게 업데이트하고 프로덕션 시간을 단축할 수 있다는 것입니다. 또한, Databricks Collaborative Notebooks를 활용하면 분야 전문가, 데이터 사이언티스트, 데이터 엔지니어가 대규모로 동일한 데이터 세트를 분석하고 노트북 환경에서 바로 협업할 수 있습니다. 그럼 시작하겠습니다!

데이터 입력 및 탐색

이 예시에서는 합성 데이터 세트를 사용할 것입니다. 데이터 세트를 직접 로드하려면 Kaggle에서 로컬 컴퓨터로 다운로드한 다음, Azure 및 AWS에서 데이터 가져오기를 통해 데이터를 가져옵니다.

PaySim 데이터는 아프리카 국가에서 구현된 휴대전화 송금 서비스의 거래 로그 1개월분에서 추출한 실제 거래 샘플을 기반으로 휴대전화 송금 거래를 시뮬레이션합니다. 아래의 표는 이 데이터 세트에서 제공하는 정보를 나타냅니다.

Column Name	Description
step	maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).
type	CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
amount	amount of the transaction in local currency.
nameOrig	customer who started the transaction
oldbalanceOrg	initial balance before the transaction
newbalanceOrig	new balance after the transaction
nameDest	customer who is the recipient of the transaction
oldbalanceDest	initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).
newbalanceDest	new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).

데이터 탐색

데이터 프레임 생성: 데이터가 Databricks File System (DBFS)에 업로드되어 있으므로 Spark SQL을 사용하여 DataFrames를 빠르고 쉽게 생성할 수 있습니다.

```
# Create df DataFrame which contains our simulated financial fraud detection dataset
df = spark.sql("select step, type, amount, nameOrig, oldbalanceOrg, newbalanceOrig,
nameDest, oldbalanceDest, newbalanceDest from sim_fin_fraud_detection")
```

DataFrame을 생성했으므로 스키마와 처음부터 1,000행까지 살펴보면서 데이터를 검토하겠습니다.

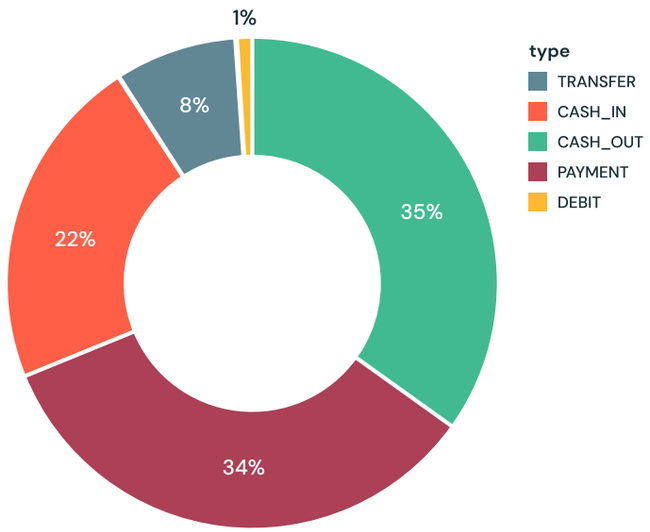
```
# Review the schema of your data
df.printSchema()
root
 |-- step: integer (nullable = true)
 |-- type: string (nullable = true)
 |-- amount: double (nullable = true)
 |-- nameOrig: string (nullable = true)
 |-- oldbalanceOrg: double (nullable = true)
 |-- newbalanceOrig: double (nullable = true)
 |-- nameDest: string (nullable = true)
 |-- oldbalanceDest: double (nullable = true)
 |-- newbalanceDest: double (nullable = true)
```

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest
1	PAYMENT	9839.64	C1231006815	170136	160296.36	M1979787155	0
1	PAYMENT	1864.28	C1666544295	21249	19384.72	M2044282225	0
1	TRANSFER	181	C1305486145	181	0	C553264065	0
1	CASH_OUT	181	C840083671	181	0	C38997010	21182
1	PAYMENT	11668.14	C2048537720	41554	29885.86	M1230701703	0
1	PAYMENT	7817.71	C90045638	53860	46042.29	M573487274	0
1	PAYMENT	7107.77	C154988899	183195	176087.23	M408069119	0
1	PAYMENT	7861.64	C1912850431	176087.23	168225.59	M633326333	0

거래 유형

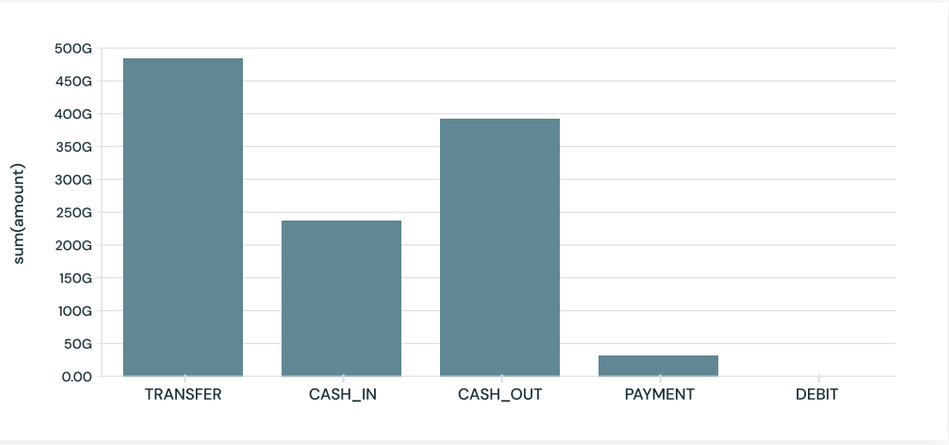
데이터를 시각화하고, 데이터가 캡처하는 거래 유형과 이 거래 유형이 전체 거래량에 미치는 영향에 대해 알아보겠습니다.

```
%sql
-- Organize by Type
select type, count(1) from financials group by type
```



운영 규모가 얼마나 되는지 알아보기 위해 거래 유형과 송금된 현금(즉, sum(amount))에 미치는 영향을 기준으로도 데이터를 시각화해보겠습니다.

```
%sql
select type, sum(amount) from financials group by type
```



규칙 기반 모델

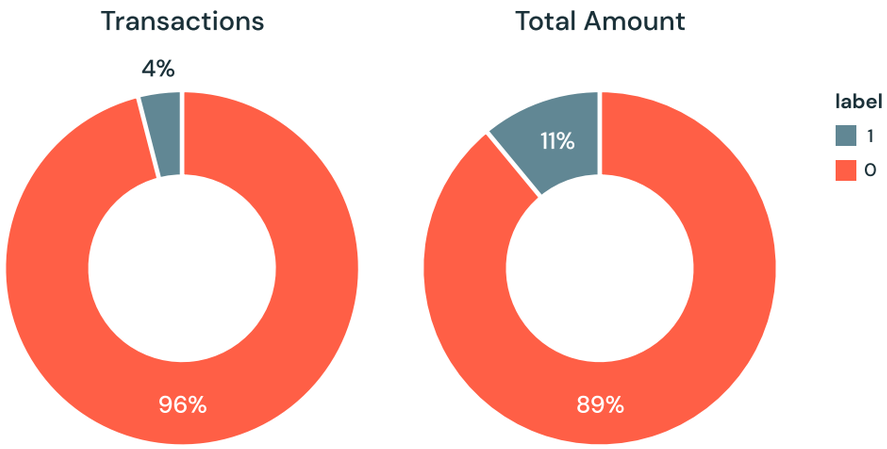
알려진 사기 사례의 데이터가 적어서 이것으로 모델을 훈련하기는 어려울 듯합니다. 이 기술을 실제로 응용하는 대부분의 경우, 분야 전문가가 설정한 규칙 세트로 사기 탐지 패턴을 찾아냅니다. 여기에서는 이 규칙에 따라 'label'이라는 열을 만들겠습니다.

```
# Rules to Identify Known Fraud-based
df = df.withColumn("label",
                   F.when(
                     (
                       (df.oldbalanceOrg <= 56900) & (df.type == "TRANSFER") & (df.
newbalanceDest <= 105)) ¦ ( (df.oldbalanceOrg > 56900) & (df.newbalanceOrig <= 12)) ¦ (
(df.oldbalanceOrg > 56900) & (df.newbalanceOrig > 12) & (df.amount > 1160000)
                             ), 1
                   ).otherwise(0))
```

규칙에서 플래그한 데이터 시각화

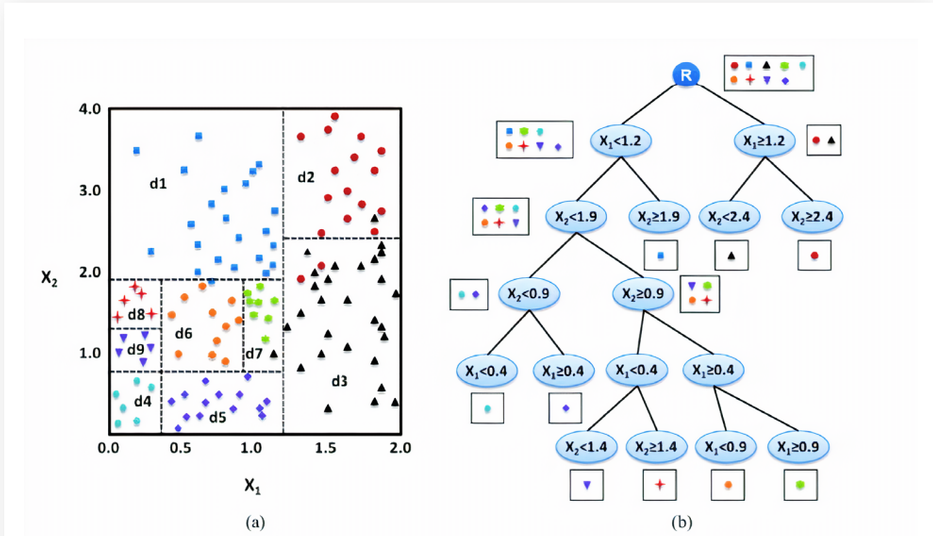
이런 규칙은 상당히 많은 사기 사례를 플래그하는 경우가 많습니다. 플래그된 거래 수를 시각화해보겠습니다. 이 규칙은 사례의 약 4%에 플래그하고, 전체 금액의 11%를 플래그합니다.

```
%sql
select label, count(1) as 'Transactions', sun(amount) as 'Total Amount' from financials_
labeled group by label
```



적절한 머신 러닝 모델 선택

대부분 사기 탐지에는 블랙박스 방식을 사용할 수 없습니다. 먼저 분야 전문가는 어떤 거래가 사기로 간주된 이유를 이해해야 합니다. 그런 다음, 법적 조치를 취한다면 증거를 법원에 제출해야 합니다. 결정 트리는 쉽게 해석할 수 있는 모델이고 이 사용 사례의 좋은 시작점입니다. 결정 트리에 대한 자세한 내용은 **"현명한 늙은 나무"** 블로그를 참조하세요.



훈련 세트 생성

ML 모델을 구축하고 검증하려면 `.randomSplit` 으로 80/20 분할이 필요합니다. 무작위로 선택한 데이터 80%를 훈련에 할당하고 나머지 20%는 결과를 검증하기 위해 남겨둡니다.

```
# Split our dataset between training and test datasets
(train, test) = df.randomSplit([0.8, 0.2], seed=12345)
```

ML 모델 파이프라인 구축

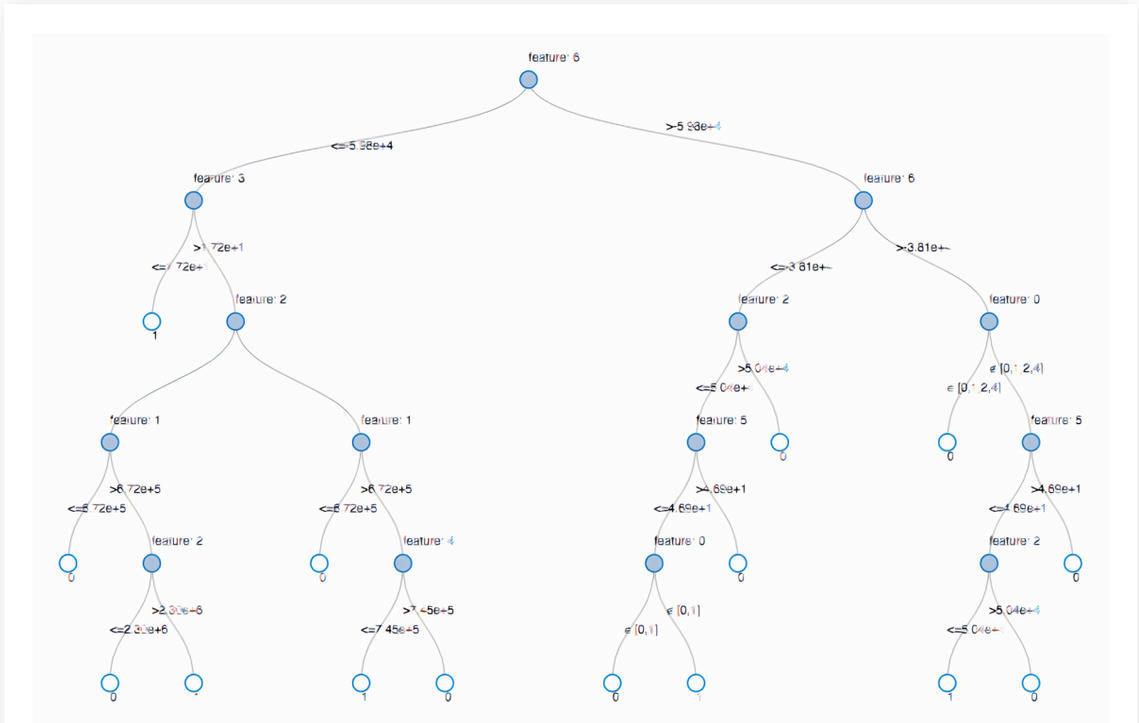
모델에 사용할 데이터를 준비하려면 먼저 카테고리 변수를 `.StringIndexer`를 사용하여 숫자로 변환해야 합니다. 그런 다음에는 모델에서 사용할 모든 feature들을 모아야 합니다. 결정 트리 모델뿐만 아니라 이런 feature 준비 단계까지 포함하는 파이프라인을 구축해야 다른 데이터 세트에서도 이 단계를 반복할 수 있습니다. 단, 훈련 데이터에 먼저 파이프라인을 피팅하고 나서 테스트 데이터를 변환해야 합니다.

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import DecisionTreeClassifier

# Encodes a string column of labels to a column of label indices
indexer = StringIndexer(inputCol = “type”, outputCol = “typeIndexed”)

# VectorAssembler is a transformer that combines a given list of columns into a single
vector column
va = VectorAssembler(inputCols = [“typeIndexed”, “amount”, “oldbalanceOrg”,
“newbalanceOrig”, “oldbalanceDest”, “newbalanceDest”, “orgDiff”, “destDiff”], outputCol =
“features”)

# Using the DecisionTree classifier model
dt = DecisionTreeClassifier(labelCol = “label”, featuresCol = “features”, seed = 54321,
maxDepth = 5)

# Create our pipeline stages
pipeline = Pipeline(stages=[indexer, va, dt])

# View the Decision Tree model (prior to CrossValidator)
dt_model = pipeline.fit(train)
```

모델 시각화

파이프라인 마지막 단계(결정 트리)에서 `display()`를 호출하면 각 노드에서 선택된 결정과 함께 초기 피팅 모델을 확인할 수 있습니다. 그러면 알고리즘이 어떻게 예측 결과에 도달했는지 이해하는 데 도움이 됩니다.

```
display(dt_model.stages[-1])
```



결정 트리 모델 그림

모델 튜닝

최적의 피팅 트리 모델을 얻기 위해서 여러 매개변수 변량으로 모델을 교차 검증할 것입니다. 우리 데이터는 음성 사례 96%와 양성 사례 4%로 구성되어 있으므로, Precision-Recall(PR) 평가 지표를 사용하여 불균형한 분포를 보정하겠습니다.

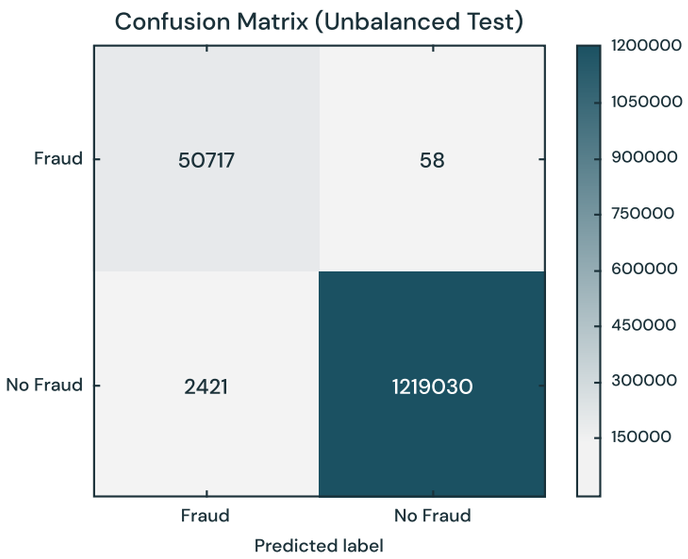
```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# Build the grid of different parameters
paramGrid = ParamGridBuilder() \
.addGrid(dt.maxDepth, [5, 10, 15]) \
.addGrid(dt.maxBins, [10, 20, 30]) \
.build()

# Build out the cross validation
crossval = CrossValidator(estimator = dt,
                          estimatorParamMaps = paramGrid,
                          evaluator = evaluatorPR,
                          numFolds = 3)
# Build the CV pipeline
pipelineCV = Pipeline(stages=[indexer, va, crossval])

# Train the model using the pipeline, parameter grid, and preceding
BinaryClassificationEvaluator
cvModel_u = pipelineCV.fit(train)
```

모델 성능

우리는 Precision-Recall(PR)과 ROC 곡선(AUC) 지표 아래의 영역을 비교하여 훈련 및 테스트 세트에 대해 모델을 평가합니다. PR과 AUC가 매우 높은 듯이 보입니다.

```
# Build the best model (training and test datasets)
train_pred = cvModel_u.transform(train)
test_pred = cvModel_u.transform(test)

# Evaluate the model on training datasets
pr_train = evaluatorPR.evaluate(train_pred)
auc_train = evaluatorAUC.evaluate(train_pred)

# Evaluate the model on test datasets
pr_test = evaluatorPR.evaluate(test_pred)
auc_test = evaluatorAUC.evaluate(test_pred)

# Print out the PR and AUC values
print(“PR train:”, pr_train)
print(“AUC train:”, auc_train)
print(“PR test:”, pr_test)
print(“AUC test:”, auc_test)

---
# Output:
# PR train: 0.9537894984523128
# AUC train: 0.998647996459481
# PR test: 0.9539170535377599
# AUC test: 0.9984378183482442
```

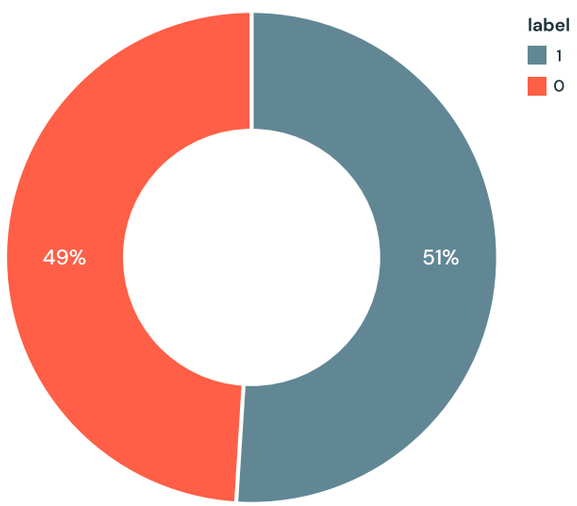
모델이 결과를 어떻게 잘못 분류하는지 알아보기 위해 Matplotlib과 pandas를 사용하여 오차 행렬을 시각화하겠습니다.



클래스 균형 조정

이 모델은 원래 규칙에서 찾아낸 것보다 2,421건 이상의 사례를 찾아냅니다. 잠재적 사기 사례를 더 많이 탐지하는 것은 긍정적일 수 있으므로 놀라지 않아도 됩니다. 그러나 원래 규칙에서 탐지되었지만 알고리즘에서 찾아내지 못한 사례가 58건 있습니다. 과소 표집을 사용하여 클래스 균형을 조정하는 방법으로 예측 결과를 개선해보겠습니다. 즉, 모든 사기 사례는 그대로 두고 사기가 아닌 사례만 과소 표집으로 조사하여 그 수치에 매칭시켜 균형 잡힌 데이터 세트를 얻는 것입니다. 새로운 데이터 세트를 시각화했을 때 사기 사례와 그렇지 않은 사례가 50/50이었습니다.

```
## Reset the DataFrames for no fraud (`dfn`) and fraud (`dfy`)
dfn = train.filter(train.label == 0)
dfy = train.filter(train.label == 1)

# Calculate summary metrics
N = train.count()
y = dfy.count()
p = y/N

# Create a more balanced training dataset
train_b = dfn.sample(False, p, seed = 92285).union(dfy)

# Print out metrics
print("Total count: %s, Fraud cases count: %s, Proportion of fraud cases: %s" % (N, y,
p))
print("Balanced training dataset count: %s" % train_b.count())

---
# Output:
# Total count: 5090394, Fraud cases count: 204865, Proportion of fraud cases:
0.040245411258932016
# Balanced training dataset count: 401898
---

# Display our more balanced training dataset
display(train_b.groupBy("label").count())
```



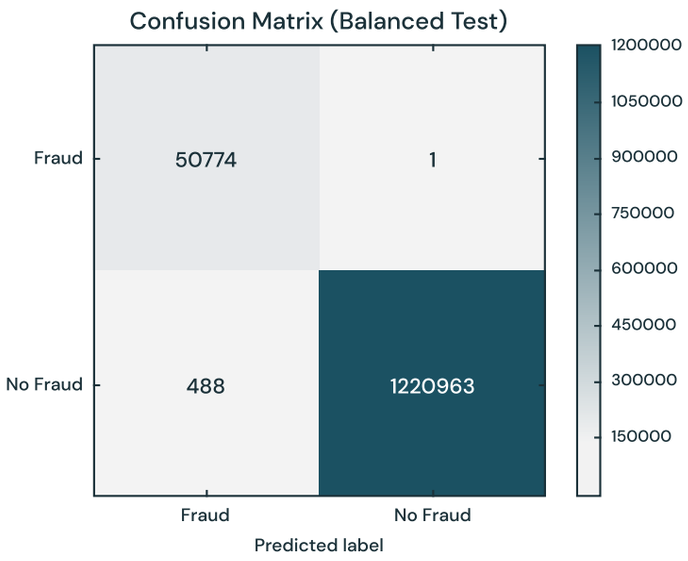
파이프라인 업데이트

이제 ML 파이프라인을 업데이트하고 새로운 교차 검증기를 생성해보겠습니다. ML 파이프라인을 사용하기 때문에 새로운 데이터 세트로 업데이트하면 동일한 파이프라인 단계를 빠르게 반복할 수 있습니다.

```
# Re-run the same ML pipeline (including parameters grid)
crossval_b = CrossValidator(estimator = dt,
estimatorParamMaps = paramGrid,
evaluator = evaluatorAUC,
numFolds = 3)
pipelineCV_b = Pipeline(stages=[indexer, va, crossval_b])

# Train the model using the pipeline, parameter grid, and
BinaryClassificationEvaluator using the `train_b` dataset
cvModel_b = pipelineCV_b.fit(train_b)

# Build the best model (balanced training and full test datasets)
train_pred_b = cvModel_b.transform(train_b)
test_pred_b = cvModel_b.transform(test)

# Evaluate the model on the balanced training datasets
pr_train_b = evaluatorPR.evaluate(train_pred_b)
auc_train_b = evaluatorAUC.evaluate(train_pred_b)

# Evaluate the model on full test datasets
pr_test_b = evaluatorPR.evaluate(test_pred_b)
auc_test_b = evaluatorAUC.evaluate(test_pred_b)

# Print out the PR and AUC values
print(“PR train:”, pr_train_b)
print(“AUC train:”, auc_train_b)
print(“PR test:”, pr_test_b)
print(“AUC test:”, auc_test_b)

---
# Output:
# PR train: 0.999629161563572
# AUC train: 0.9998071389056655
# PR test: 0.9904709171789063
# AUC test: 0.9997903902204509
```

결과 검토

새로운 오차 행렬의 결과를 살펴보겠습니다. 이 모델은 사기 사례 1개만 잘못 판정했습니다. 클래스 균형을 조정한 덕분에 모델이 개선된 듯합니다.



모델 피드백 및 MLflow 사용

프로덕션에 사용할 모델을 선택했다면 모델이 지속해서 찾으려는 행위를 찾아낼 수 있도록 꾸준하게 피드백을 수집해야 합니다. 규칙 기반 레이블로 시작했기 때문에 인간의 피드백을 받아 검증한 실제 레이블을 포함한 추가적인 모델을 제공하고자 합니다. 이 단계는 머신 러닝 프로세스에 대한 신뢰와 확신을 유지하는 데 중요합니다. 분석 전문가가 모든 사례를 검토할 수 없으므로, 모델 결과값을 검증하기 위한 사례를 신중히 선택하여 제공하고자 합니다. 예를 들어 모델의 신뢰도가 낮은 예측은 분석 전문가가 검토하면 좋습니다. 이런 피드백이 있으면 모델을 변화하는 환경에 맞춰 지속해서 개선할 수 있습니다.

MLflow는 이 사이클에서 다양한 모델 버전을 훈련하도록 지원합니다. 실험을 추적하면서 각 모델 구성과 매개변수의 결과를 비교할 수 있습니다. 예를 들어 MLflow UI를 사용하여 균형 잡힌 데이터 세트와 불균형한 데이터 세트에서 훈련한 모델의 PR과 AUC를 비교할 수 있습니다. 데이터 사이언티스트는 MLflow를 사용해서 다양한 모델 지표와 추가적 시각화, 아티팩트를 추적하고 프로덕션에서 어느 모델을 배포할지 결정하는 데 도움을 받을 수 있습니다. 데이터 엔지니어는 선택한 모델과 훈련에 사용할 라이브러리 버전을 프로덕션의 새로운 데이터에 배포할 .jar 파일로 쉽게 가져올 수 있습니다. 따라서 모델 결과를 검토하는 분야 전문가, 모델을 업데이트하는 데이터 사이언티스트, 프로덕션에 모델을 배포하는 데이터 엔지니어는 이런 반복적 과정을 통해 협업이 강화됩니다.

결론

Databricks와 MLflow를 사용하여 규칙 기반 사기 탐지 레이블을 머신 러닝 모델로 변환하는 방법의 예시를 살펴보았습니다. 이 방법을 사용하면 확장할 수 있는 모듈식 솔루션을 개발해, 끊임없이 변화하는 사기 행위 패턴을 따라가는 데 도움이 됩니다. 머신 러닝 모델을 구축하여 사기를 찾아내면 모델을 발전시키고 새로운 잠재적 사기 패턴을 찾아내는 피드백 루프를 구축할 수 있습니다. 특히, 결정 트리 모델은 상호운용성과 정확도가 우수해서 사기 탐지 프로그램에 머신 러닝을 도입하기에 좋은 시작점이 될 수 있다는 것을 확인했습니다.

여기에 Databricks 플랫폼을 사용하는 가장 큰 장점은 데이터 사이언티스트, 엔지니어, 비즈니스 사용자가 프로세스 전반에서 매끄럽게 협력할 수 있다는 것입니다. 데이터 준비, 모델 구축, 결과 공유, 모델을 프로덕션에 배포하는 작업을 모두 동일한 플랫폼에서 처리할 수 있으므로 지금까지와는 차원이 다른 협업을 경험할 수 있습니다. 이 방법은 사일로화되어 있던 팀에서 신뢰를 키우고 동적이고 효과적인 사기 탐지 프로그램을 만들 수 있습니다.

몇 분만 투자해서 무료 체험에 등록하여 **이 노트북**을 체험해보고 자신의 모델을 만들어 보세요.

6장:

Databricks에서 머신 러닝으로 디지털 병리학 이미지 분석 자동화

이미지 촬영 기술이 발전하고 새로운 효율적 컴퓨팅 도구가 출시되면서 디지털 병리학이 연구 진단 분야에서 주목받고 있습니다. 이런 혁신의 중심에는 WSI(Whole Slide Imaging)가 있고, 병리학 슬라이드를 고해상도 이미지로 빠르게 디지털화하는 데 도움을 주고 있습니다. WSI를 사용하면 이미지를 즉시 공유하고 분석할 수 있기 때문에 재현성이 개선되었고 교육 및 원격 병리 서비스도 더욱 개선할 수 있었습니다.

현재 초고해상도로 전체 슬라이드를 디지털화하는 작업은 1분 이내로 쉽게 완료됩니다. 따라서 방대한 디지털 슬라이드 카탈로그를 구매하는 의료 기관과 생명공학 기관이 점점 늘어나고 있습니다. 이런 대규모 세트는 머신 러닝을 적용한 자동 진단을 구축하는 데 사용할 수 있으며, 슬라이드나 그 세그먼트를 분류하여 특정 표현형으로 표현하거나 슬라이드에서 정량적 생물 지표를 직접 추출할 수 있습니다. 머신 러닝과 딥러닝을 활용하면 수천 개의 디지털 슬라이드를 몇 분 만에 해석할 수 있습니다. 이는 병리학과, 임상의 연구자가 암과 감염성 질병을 진단하고 치료하는 효율과 능률을 개선할 절호의 기회입니다.

글: **Amir Kermany** and **Frank Austin Nothaft**

2020년 1월 31일

Databricks에서 노트북 보기 →

databricks

디지털 병리학 워크플로의 광범위한 도입을 막는 3가지 공통적 문제

많은 의료 기관과 생명공학 기관이 인공 지능을 모든 슬라이드 이미지에 적용했을 때의 잠재적 효과는 인정하지만 자동 슬라이드 분석 파이프라인을 도입하는 데는 아직 복잡한 과정이 있습니다. 운영 WSI 파이프라인은 저렴한 비용으로 디지털 슬라이드를 정기적으로 대량 처리할 수 있어야 합니다. 데이터 사이언스로 지원하는 자동 디지털 병리학 워크플로를 구현하지 못하는 공통적 원인은 세 가지가 있습니다.

1. **느리고 값비싼 데이터 입력 및 엔지니어링 파이프라인:** 일반적으로 WSI 이미지는 매우 크고 (일반적으로 슬라이드당 0.5~2GB) 광범위한 이미지 처리가 필요합니다.

2. **딥러닝을 테라바이트 규모 이미지에 확장하지 못하는 문제:** 수백 개의 WSI가 포함된 소량의 데이터 세트로 딥러닝 모델을 하나의 노드에서 훈련하는 데 며칠, 심지어 몇 주가 걸릴 수도 있습니다. 이런 지연이 발생하면 대량의 데이터 세트에 대해 신속히 실험을 진행할 수 없습니다. 여러 노드에서 딥러닝 워크로드를 나란히 처리하여 지연을 줄일 수 있지만, 이는 일반적 생물학 데이터 사이언티스트는 바랄 수 없는 고급 기술입니다.

3. **WSI 워크플로의 재현성:** 환자 데이터에서 새로운 인사이트를 얻을 때는 결과를 재현하는 것이 매우 중요합니다. 기존의 솔루션은 대부분 임시에 불과하고, 머신 러닝 모델 훈련에서 사용한 데이터 실험과 버전을 효율적으로 추적할 수 없습니다.

이 블로그에서는 Databricks Unified Data Analytics Platform을 사용하여 이러한 문제를 해결하고 WSI 이미지 데이터에서 확장할 수 있는 전체적 딥러닝 워크플로를 배포해보겠습니다. 슬라이드에서 전이 부위를 찾아내는 이미지 세그먼테이션 모델을 훈련하는 워크플로를 집중적으로 다룰 것입니다. 이 예시에서는 Apache Spark를 사용하여 이미지 모음에 대해 데이터를 동시에 처리하고, pandas UDF를 사용하여 여러 노드에서 사전 훈련된 모델(전이 학습)을 기반으로 feature들을 추출한 뒤, MLflow를 사용하여 모델 훈련 추적을 재현해보겠습니다.

WSI에서 머신 러닝의 전 과정 처리

Databricks 플랫폼을 사용하여 WSI 데이터 처리 파이프라인을 가속화하는 방법을 보여드리기 위해 **Camelyon16 Grand Challenge 데이터 세트**를 사용할 것입니다. 유방암 조직에서 얻은 **TIFF 형식**의 슬라이드 이미지 400개로 구성된 공개 액세스 데이터 세트로 워크플로를 보여드리겠습니다. Camelyon16 데이터 세트의 하위 집합은 /databricks-datasets/med-images/camelyon16/(AWS | Azure)의 Databricks에서 직접 액세스할 수 있습니다. 슬라이드에서 암 전이가 있는 부위를 찾아내는 이미지 분류기를 훈련하기 위해 그림 1과 같이 3단계를 실행합니다.

1. **패치 생성:** 병리학자가 기록한 좌표를 사용하여 슬라이드 이미지를 동일한 크기의 패치로 자릅니다. 각 이미지는 수천 개의 패치를 생성할 수 있으며 종양 또는 정상으로 레이블이 표시됩니다.
2. **딥러닝:** 전이 학습으로 사전 훈련된 모델을 사용하여 이미지 패치에서 feature를 추출한 다음, Apache Spark를 사용하여 종양과 정상 패치를 예측하는 바이너리 분류자를 훈련합니다.
3. **평가:** 그런 다음, MLflow를 사용하여 기록한 훈련된 모델로 특정 슬라이드의 확률 열지도를 도출합니다.

방사선 이미지 전처리에 사용한 Human Longevity 워크플로와 마찬가지로, Apache Spark를 사용하여 슬라이드와 주석을 모두 조정합니다. 모델 훈련의 경우, Keras에서 사전 훈련된 **InceptionV3** 모델을 사용하여 feature를 추출하는 것으로 시작합니다. 이를 위해 **pandas UDF**를 사용하여 feature 추출을 동시에 처리합니다. 이 기술에 대한 자세한 내용은 전이 학습을 위한 featurization(**AWS** | **Azure**)를 참조하세요. 이 기술은 InceptionV3에만 국한되지 않고 다른 사전 훈련된 모델에도 적용할 수 있습니다.

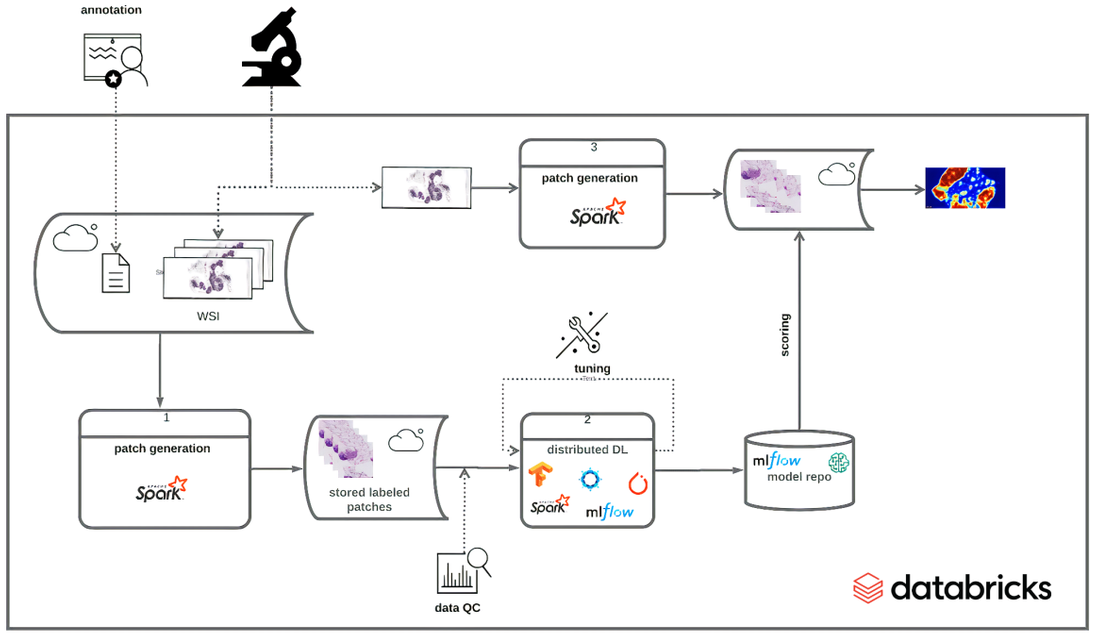


그림 1: WSI 데이터에 기반한 DL 모델을 훈련, 배포하기 위한 전체적 솔루션 구현

이미지 전처리 및 ETL

병리학자는 Automated Slide Analysis Platform와 같은 오픈 소스 도구를 사용하여 매우 세밀하게 WSI 이미지를 검토하고, 임상적으로 중요한 부위를 슬라이드에 표시할 수 있습니다. 이 주석은 XML 파일로 저장되며, 해당 부위와 다른 정보(예: 확대 수준)를 포함한 폴리곤의 가장자리 좌표가 들어갑니다. 실측 슬라이드 세트에 대한 주석을 사용하는 모델을 훈련하려면 이미지별로 주석이 기록된 부위의 목록을 로드하고 이 부위와 이미지를 결합하여, 주석이 있는 부위를 잘라내야 합니다. 이 과정이 끝나면 이미지 패치를 머신 러닝에 사용할 수 있습니다.

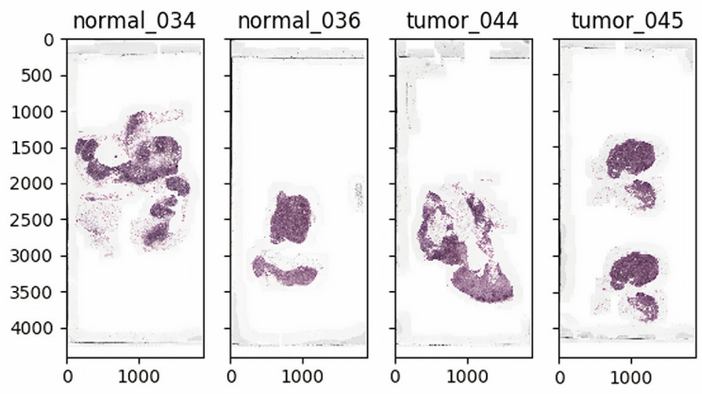


그림 2: Databricks 노트북에서 WSI 이미지 시각화

이 워크플로는 XML 파일에 저장된 주석에 사용하는 것이 일반적이지만 간단하게 처리하기 위해 Camelyon 16 데이터 세트에 대한 NCRF 분류자를 개발한 Baidu 연구팀의 전처리된 주석을 사용하겠습니다. 이들 주석은 CSV 인코딩 텍스트 파일로 저장되고 Apache Spark는 DataFrame으로 로드합니다. 다음의 노트북 셀에서 종양과 정상 패치에 대한 주석을 로드하고, 정상 슬라이드에는 0을, 종양 슬라이드에는 1을 레이블로 표시합니다. 그런 다음, 좌표와 레이블을 하나의 DataFrame으로 통합합니다.

대부분 SQL 기반 시스템에서는 내장된 작업만 지원하지만, Apache Spark는 사용자 정의 함수(UDF)를 다양하게 지원합니다. UDF를 사용하면 Apache Spark DataFrame에서 데이터에 사용자 정의 Scala, Java, Python, R 함수를 호출할 수 있습니다. 우리 워크플로에서는 OpenSlide 라이브러리를 사용하는 Python UDF를 정의하여 이미지에서 특정 패치를 잘라냅니다. 처리할 WSI의 이름, 패치 중심의 X와 Y 좌표, 패치 레이블을 을 받는 Python 함수를 정의하고, 나중에 훈련에 사용할 타일을 생성합니다.

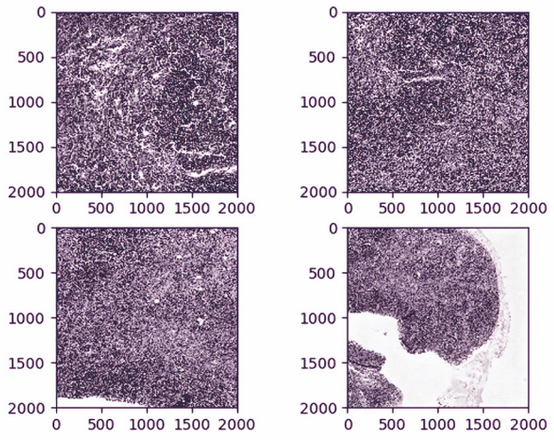


그림 3: 확대 수준별 패치 시각화

이제 OpenSlide 라이브러리를 사용하여 클라우드 스토리지에서 이미지를 로드하고 특정 좌표 범위로 잘라냅니다. OpenSlide는 Amazon S3 또는 Azure Data Lake Storage에서 데이터를 읽는 기능을 기본적으로 제공하지는 않지만, Databricks File System(DBFS) FUSE 계층에서 복잡한 코드 변경 없이 blob 스토어에 저장된 데이터에 직접 액세스할 수 있도록 지원합니다. 마지막으로, DBFS FUSE 계층을 사용하여 함수에서 패치를 다시 작성합니다.

이 명령이 Databricks 데이터 세트에 대해 Camelyon16 데이터 세트에서 최대 174,000개의 패치를 생성하는 데 약 10분이 소요됩니다. 명령이 완료되면 패치를 다시 로드하고 노트북에서 인라인으로 직접 표시할 수 있습니다.

전이 학습과 MLflow를 사용한 종양/정상 병리학 분류자 훈련

앞의 단계에서는 패치와 관련 메타데이터를 생성하고, 클라우드 스토리지에 생성된 이미지 타일을 저장했습니다. 이제 바이너리 분류자를 훈련하여 슬라이드 세그먼트에 종양 전이가 있는지 예측할 준비를 할 것입니다. 이를 위해서 전이 학습을 사용하여 사전 훈련된 **딥 신경망**으로 각 패치에서 feature를 추출하고, sparkml을 분류 작업에 적용합니다. 대부분 이미지 처리 분야에서는 처음부터 훈련을 시작하는 것보다 이 기술을 사용하는 것이 더 높은 성과를 내는 경우가 많습니다. 먼저 InceptionV3 아키텍처에서 Keras의 사전 훈련된 가중치를 사용해보겠습니다.

Apache Spark의 DataFrames는 이미지 스키마가 내장되어 있고 모든 패치를 DataFrame으로 직접 로드할 수 있습니다. 그런 다음, Pandas UDF를 사용하여 Keras를 사용하는 InceptionV3를 기반으로 이미지를 feature로 변환합니다. 각 이미지를 특징화했다면 **spark.ml**을 사용하여 각 패치의 feature와 레이블 사이의 로지스틱 회귀를 피팅합니다. MLflow로 로지스틱 회귀 모델을 기록하면 나중에 모델에 액세스할 수 있습니다.

Databricks에서 ML 워크플로를 실행할 때 관리형 MLflow를 사용할 수 있습니다. 각 노트북 런과 훈련 라운드에서 MLflow가 매개변수, 지표, 지정된 아티팩트를 자동으로 로깅합니다. 또한, 나중에 데이터에서 레이블을 예측하는 데 사용할 훈련 모델도 저장합니다. MLflow를 사용하여 Databricks의 ML 워크플로 전체 사이클을 관리하는 방법을 자세히 알아보고 싶은 독자 여러분께서는 **이 문서**를 참조하세요.

워크플로	시간
패치 생성	10분
Feature 엔지니어링 및 훈련	25분
평가(슬라이드 1개당)	15초

표 1: Databricks ML Runtime 6.2를 사용한 2-10 r4.4xlarge 작업자로 워크플로의 각 단계를 실행하는 시간, databricks-datasets에 포함된 슬라이드에서 추출한 170,000개 패치

표 1은 워크플로의 각 부분에서 소요된 시간을 나타냅니다. 최대 17만 개의 샘플로 모델을 훈련하는 데 걸리는 시간은 25분 미만이고 정확도는 87%입니다.

실제로는 패치가 더 많을 수 있기 때문에 분류에 딥 신경망을 사용하면 정확도가 상당히 개선됩니다. 이와 같은 경우, 분산된 훈련 기술을 사용하여 훈련 과정을 확장할 수 있습니다. Databricks 플랫폼에는 **HorovodRunner** 툴킷 패키지를 제공하는데, 이는 ML 코드를 아주 약간 수정하는 것만으로 대용량 클러스터에 훈련 작업을 배포할 수 있습니다. **이 블로그 게시물**은 Databricks에서 ML 워크플로를 확장하는 방법에 대한 자세한 설명을 제공합니다.

추론

분류자 훈련이 끝나면 이를 사용하여 슬라이드에 전이가 있을 확률을 열지도로 나타내야 합니다. 이를 위해서는 먼저 슬라이드에서 원하는 세그먼트에 그리드를 적용한 다음, 훈련 과정과 유사한 방법으로 패치를 생성해서 Spark DataFrame으로 데이터를 가져옵니다. 이 데이터는 나중에 예측에 사용할 수 있습니다. 그런 다음, MLflow를 사용하여 훈련된 모델을 로드하고 예측을 연산하는 DataFrame에 변환해서 적용합니다.

이미지를 재구성하기 위해 Python의 PIL 라이브러리를 사용하여 전이 부위가 포함될 확률에 따라 각 타일 색상을 수정하고 모든 타일을 함께 연결합니다. 아래의 그림 4는 종양 세그먼트 중 하나에 대한 확률 예측 결과를 나타냅니다. 빨간색 밀도는 슬라이드에 전이가 있을 확률이 높다는 것을 나타냅니다.

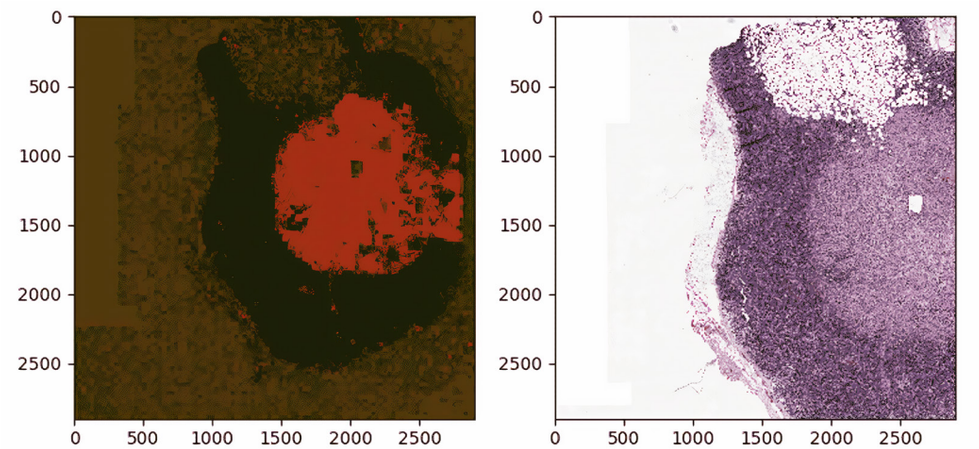


그림 4: WSI의 특정 세그먼트에 예측 매핑

병리학 이미지에 머신 러닝 적용

이 블로그에서는 Databricks에서 Spark SQL, SparkML, MLflow를 함께 사용하여 병리학 이미지에 머신 러닝을 적용하기 위한 재현과 확장할 수 있는 프레임워크를 구축하는 방법을 보여드립니다. 특히, 대규모 전이 학습으로 슬라이드 세그먼트에 암 세포가 있을 확률을 예측하는 분류자를 훈련한 다음, 훈련된 모델을 사용하여 특정 슬라이드에 암종을 찾아내 매핑했습니다.

먼저 **Databricks 무료 체험**에 등록하고 **WSI 이미지 세그먼테이션** 노트북으로 실험해보세요. **의료** 및 **생명공학** 페이지에서 다른 솔루션에 대해서도 알아보실 수 있습니다.

7장:

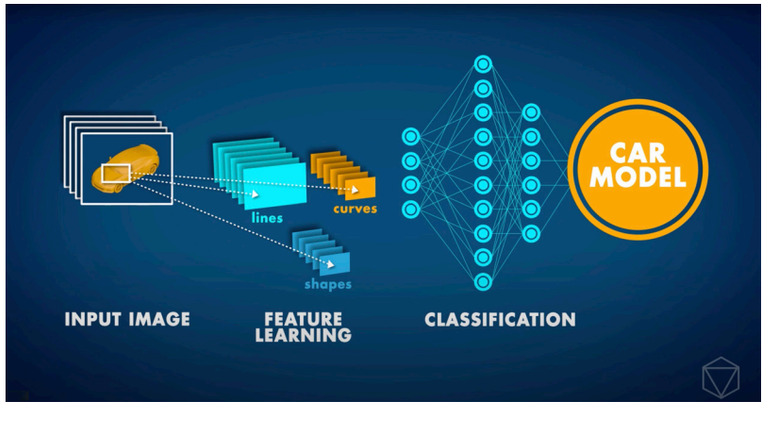
자동차 분류를 위한 컨볼루션 신경망 구현

글: Dr. Evan Eames 및 Henning Kropp

2020년 5월 14일

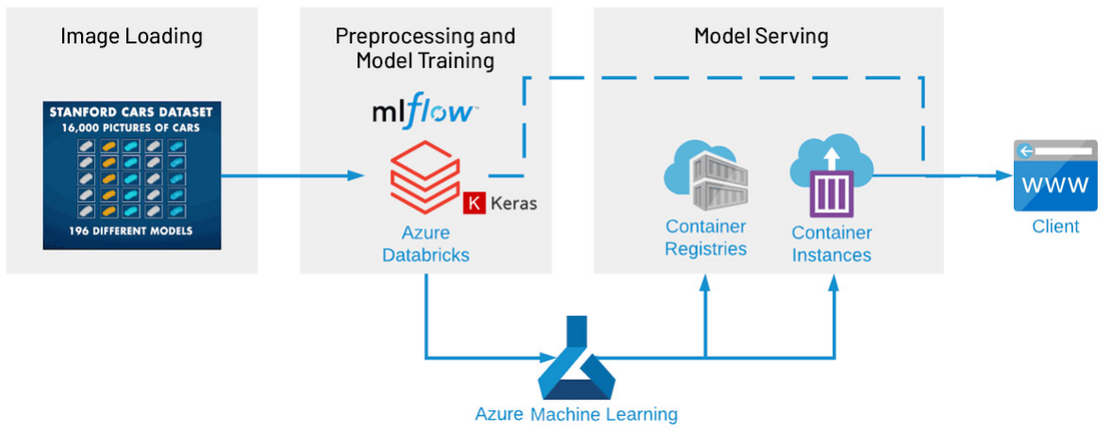
컨볼루션 신경망(CNN)은 주로 컴퓨터 비전 작업에 사용하는 첨단 신경망 아키텍처입니다. CNN은 이미지 인식, 개체 현지화, 변경 탐지 등의 다양한 작업에 적용할 수 있습니다. 얼마 전 우리 파트너 **Data Insights**는 주요 자동차 제조사로부터 까다로운 요청을 받았습니다. 바로, 특정 이미지에서 자동차 모델을 식별하는 컴퓨터 비전 애플리케이션을 개발해달라는 요청이었습니다. 다른 자동차 모델이라도 상당히 유사하게 보일 수 있고 주변 환경과 사진을 촬영하는 각도에 따라 같은 자동차라도 매우 다르게 보일 수 있다는 점을 고려하면, 이런 작업은 불과 얼마 전까지도 해결할 수 없었습니다.

그러나 2012년경부터 `딥러닝의= 혁신`으로 인해 이런 문제를 처리할 수 있게 되었습니다. 컴퓨터는 자동차의 콘셉트를 입력받지 않아도, 반복적으로 사진을 검사해서 자동차의 콘셉트를 직접 학습할 수 있게 되었습니다. 최근 몇 년 사이에 인공 신경망(ANN)이 더욱 혁신적으로 발전한 덕분에 인간 수준의 정확도로 이미지를 분류할 수 있는 AI가 나타나게 되었습니다. 이런 기술 발전을 바탕으로 Deep CNN이 자동차 모델을 분류하도록 훈련할 수 있게 되었습니다. 신경망은 Stanford Cars Data Set로 훈련하였으며, 여기에는 196개 모델로 구성된 자동차 사진 16,000개 이상이 포함되어 있습니다. 신경망이 자동차 콘셉트와 각 모델을 구분하는 법을 학습하는 동안 시간이 지남에 따라 예측의 정확도가 향상되기 시작했습니다.



입력과 출력 계층 사이에 여러 계층이 존재하는 인공 신경망(ANN)의 예시입니다. 이미지를 입력하면 자동차 모델 분류 결과가 출력됩니다.

우리는 파트너와 협력하여 머신 러닝 파이프라인 전체를 구축했습니다. 데이터 전처리에는 Apache Spark™와 Koalas를 사용하였고, 모델 훈련에는 Keras와 TensorFlow를 사용하였으며, 모델 및 결과 추적에는 MLflow를 사용하였고, REST 서비스 배포에는 Azure ML을 사용했습니다. Azure Databricks 내의 설정은 빠르고 효율적으로 네트워크를 훈련하도록 최적화했으며, 다른 여러 가지 CNN 구성도 훨씬 빠르게 시도하는 데 도움이 됩니다. 몇 번의 연습 시도만으로 CNN의 정확도는 약 85%에 도달했습니다.



이미지를 분류하기 위한 인공 신경망(ANN) 설정

이 글에서는 신경망을 프로덕션에 배포하기 위해 사용한 주요 기술 몇 가지를 간략히 설명합니다. 신경망을 직접 운영하고 싶을 분을 위해 자세한 단계별 가이드가 포함된 전체 노트북을 아래에서 제공합니다.

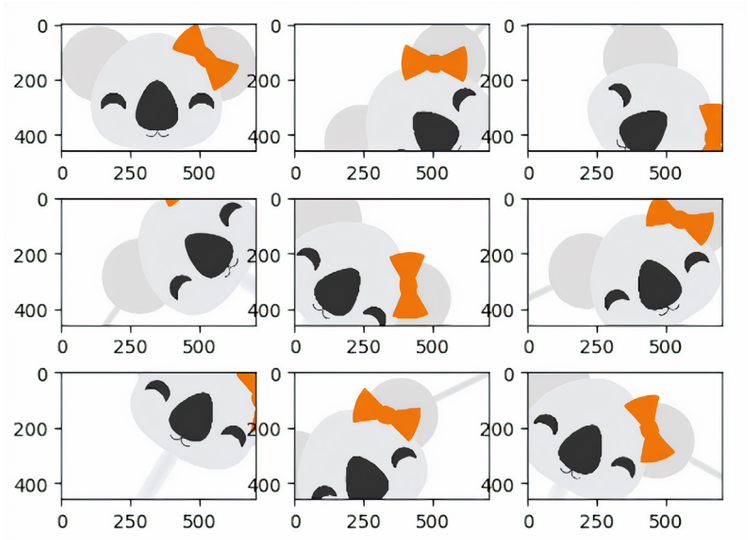
이 데모는 공개된 Stanford Cars Data Set를 사용합니다. 이 데이터 세트는 가장 종합적인 공개 데이터 세트 중 하나이지만, 다소 오래되어서 2012년 이후의 차종은 찾을 수 없습니다(하지만 훈련이 끝나면 전이 학습으로 손쉽게 새로운 데이터 세트로 대체할 수 있습니다). 이 데이터는 작업 영역에 마운트할 수 있는 ADLS Gen2 스토리지 계정을 통해 제공됩니다.

데이터 전처리의 첫 단계에서 이미지는 hdf5 파일로 압축합니다(하나는 훈련용, 다른 하나는 테스트용). 이 파일은 신경망에서 읽을 수 있습니다. hdf5 파일은 이 노트북에서 제공하는 ADLS Gen2 스토리지에 포함되어 있으므로, 원하신다면 이 단계를 완전히 생략해도 됩니다.

- Stanford Cars 데이터 세트를 HDF5 파일에 로드
- Koalas를 이미지 보강에 사용
- Keras로 CNN 훈련
- Azure ML에 REST 서비스로 모델 배포

Koalas로 이미지 보강

수집된 데이터의 수량과 다양성은 딥러닝 모델로 얻을 수 있는 결과에 큰 영향을 미칩니다. 데이터 보강은 새 데이터를 실제로 수집하지 않고도 학습 결과를 상당히 개선할 수 있는 전략입니다. 자르기, 패딩, 수평 회전 등의 다양한 기술은 대규모 신경망을 훈련하는 데 널리 사용되며, 이를 사용하여 데이터 세트를 인위적으로 부풀려 훈련과 테스트에 사용할 이미지 수를 늘릴 수 있습니다.



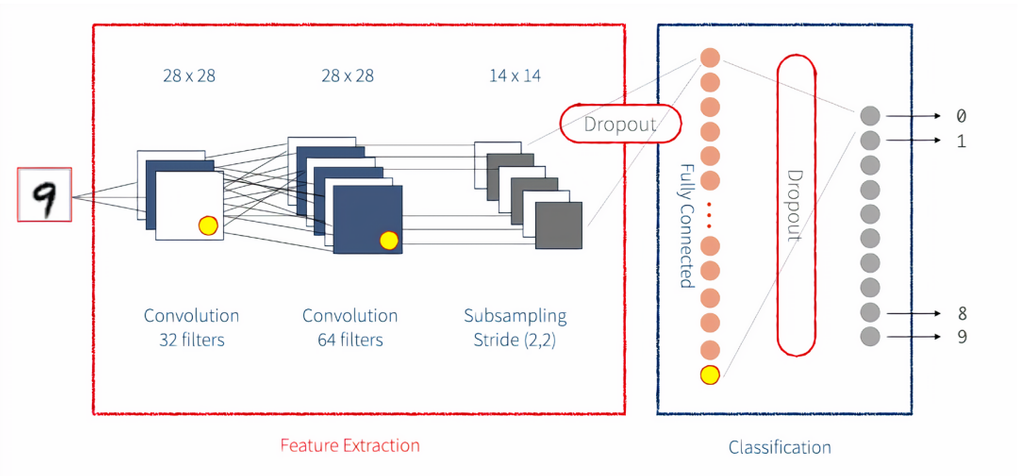
대량의 훈련 데이터를 보강하는 데는 엄청난 비용이 들 수 있습니다. 특히, 여러 방식을 적용한 결과를 비교할 때는 더욱 그렇습니다. Koalas를 사용하면 기존 프레임워크에서 Python으로 이미지 보강을 쉽게 시도할 수 있고, pandas API와 유사한 데이터 사이언스를 사용하여 여러 노드가 있는 클러스터로 프로세스를 확장할 수도 있습니다.

Keras로 ResNet 코딩

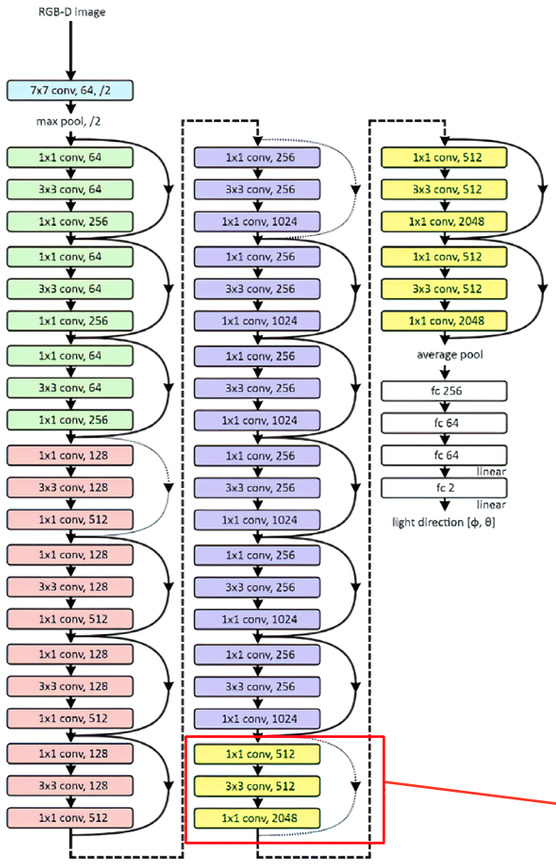
CNN을 분석해보면 다양한 '블록'으로 구성된 것을 알 수 있습니다. 각 블록은 일부 입력 데이터에 적용할 작업을 모아둔 것에 불과합니다. 이런 블록은 크게 다음과 같이 분류할 수 있습니다.

- **ID 블록:** 데이터의 형태를 동일하게 유지하는 일련의 작업
- **컨볼루션 블록:** 입력 데이터의 형태를 작은 형태로 압축하는 일련의 작업

CNN은 ID 블록과 컨볼루션 블록(ConvBlock)이 모두 존재하고, 입력한 이미지를 간단한 숫자들로 압축합니다. 이렇게 얻은 각각의 숫자는 (올바르게 훈련한다면) 이미지 분류에 대해 유용한 정보를 전달합니다. 잔존 CNN은 각 블록에 추가적 단계를 더합니다. 데이터는 임시 변수로 저장되었다가 블록을 구성하는 작업을 적용하고, 이 임시 데이터를 출력 데이터에 추가합니다. 일반적으로 이런 추가적 단계는 각 블록에 적용됩니다. 예를 들어, 아래의 그림은 손으로 쓴 숫자를 탐지하기 위한 간단한 CNN을 나타냅니다.



신경망을 구현하는 방법은 여러 가지가 있습니다. Keras를 사용하는 방법이 직관적인 편입니다. Keras는 신경망을 구성하는 각 단계를 실행하는 간단한 프런트엔드 라이브러리를 제공합니다. Keras는 Tensorflow 백엔드 또는 Theano 백엔드로 작업할 수 있도록 구성할 수 있습니다. 여기에서는 TensorFlow 백엔드를 사용할 것입니다. Keras 네트워크는 아래와 같이 여러 계층으로 나뉩니다. 우리 네트워크에서는 계층의 고객 구현도 정의합니다.



네트워크는 224 x 224 x 4 크기의 입력 이미지로 시작합니다. 4가지 차원은 RGB-D 이미지 채널을 나타냅니다.

입력 계층 다음에는 컨볼루션 계층이 이어지고, 커널은 64개이며 이미지 크기는 7 x 7입니다. 또한, 이 계층은 이미지 크기를 절반으로 나누는 데도 사용합니다(/2).

이 네트워크는 최대 풀링 계층을 적용하고, 다시 해상도를 반으로 나눕니다.

그러면 네트워크는 16개의 잔존 블록으로 구성된 컨볼루션 계층 48개가 존재합니다. 이 잔존 블록에는 증식하는 커널이 있습니다.

컨볼루션 계층 다음에는 평균 풀링이 이어지고, 4개의 완전히 연결된 계층(감소하는 뉴런)이 이어집니다.

마지막 계층은 회귀합니다.

16개 잔존 블록 중 1개이고 각각 컨볼루션 계층은 3개입니다.

확장 계층

훈련할 수 있는 가중치가 있는 사용자 정의 작업의 경우, Keras를 사용하여 **자체적 계층을 구현**할 수 있습니다. 대량의 데이터를 처리할 때는 메모리 문제가 발생할 수 있습니다. 본래 RGB 이미지에는 정수 데이터(0-255)가 포함됩니다. 역전달 중에 최적화를 진행하면서 기울기 하강을 실행하면 이런 정수 기울기값으로 인해 네트워크 가중치를 적절히 조정할 정도의 정확도에 도달할 수 없다는 것을 알게 됩니다. 그러므로 부동 소수점 정밀도로 변경해야 합니다. 이 과정에서 문제가 발생할 수 있습니다. 이미지를 224 x 224 x 3으로 축소하더라도 훈련 이미지가 10,000개이면 부동 소수점 항목이 10억 개 이상이 됩니다. 모든 데이터 세트를 부동 소수점 정밀도로 바꾸는 대신 “확장 계층(Scale layer)”을 사용하여 필요한 경우에만 이미지 하나씩 입력 데이터를 확장하는 방법이 유리합니다. 이는 모델에서 일괄 정규화를 처리한 후에 적용해야 합니다. 이 확장 계층의 매개변수는 훈련을 통해 학습할 수 있는 매개변수이기도 합니다.

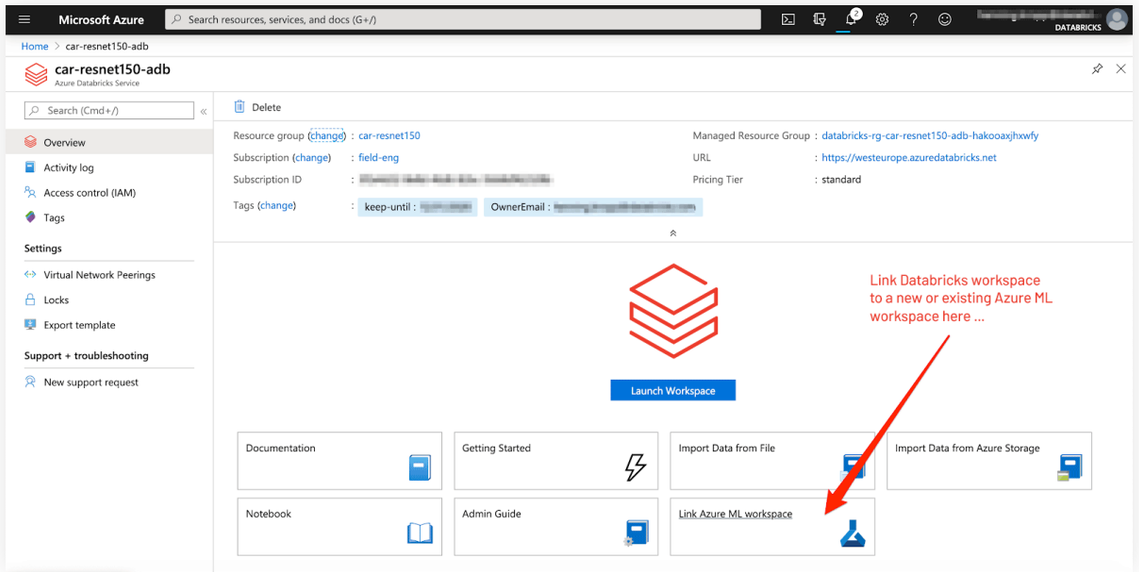
평가 중에도 이 사용자 정의 계층을 사용하려면 모델에 이 클래스를 함께 패키징해야 합니다. MLflow에서는 Keras 모델과 관련된 사용자 정의 클래스 또는 함수에 대한 Keras custom_objects 사전 매핑 이름(문자열)을 사용하면 됩니다. MLflow는 CloudPickle로 이런 사용자 정의 계층을 저장하고 모델을 `mlflow.keras.load_model()` 과 `mlflow.pyfunc.load_model()` 로 로드했을 때 자동 복구됩니다.

```
mlflow.keras.log_model(model, “model”, custom_objects={“Scale”: Scale})
```

MLflow 및 Azure Machine Learning으로 결과 추적

머신 러닝 배포는 소프트웨어 개발에서 복잡성이 더해지는 원인입니다. 수많은 도구와 프레임워크가 있기 때문에 실험 추적, 결과 재현, 머신 러닝 모델 배포가 어렵습니다. Azure Machine Learning을 사용하면 MLflow를 사용하여 전체적 머신 러닝 수명 주기를 단축하여 관리하면서 Azure Databricks를 사용하여 머신 러닝 애플리케이션을 안정적으로 구축, 공유, 배포할 수 있습니다.

결과를 자동으로 추적하기 위해 기존 또는 새로운 Azure ML 작업 영역을 Azure Databricks 작업 영역을 연결할 수 있습니다. 또한, MLflow는 Keras 모델(mlfl w.keras.autolog())에 자동 로깅을 지원하여 이 과정을 손쉽게 처리할 수 있습니다.

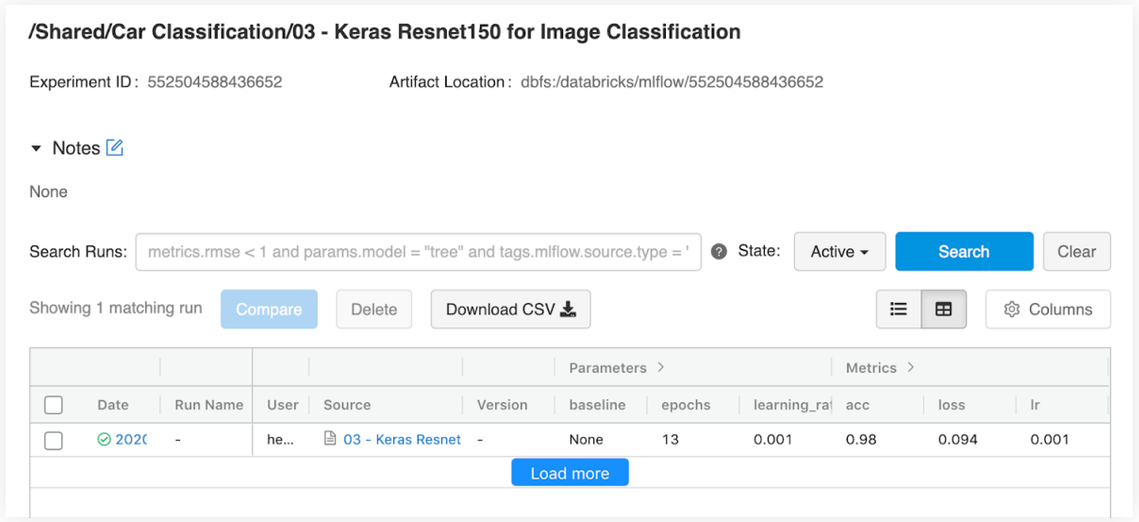


MLflow의 내장 모델 영구 유틸리티는 Keras 등의 여러 가지 일반적인 ML 라이브러리에서 모델을 패키징하는 데 편리하게 사용할 수 있지만, 모든 사용 사례에 적용할 수는 없습니다. 예를 들어, MLflow의 내장 플레이버에서 명시적으로 지원하지 않는 ML 라이브러리에서 모델을 사용해야 할 수도 있습니다. 또는, 사용자 정의 추론 코드와 데이터를 패키징해서 MLflow 모델을 생성할 수도 있습니다. 다행히 MLflow는 이 작업을 처리하기 위한 솔루션을 두 가지 제공합니다. 바로 Custom Python Models와 Custom Flavors입니다.

이 시나리오에서는 REST API 클라이언트에서 요청을 보낼 수 있는 모델 추론 엔진을 사용하고자 합니다. 이를 위해서는 앞서 구축한 Keras 모델을 기반으로 사용자 정의 모델을 사용하여 Base64 인코딩 이미지가 들어 있는 JSON DataFrame 개체를 입력해야 합니다.

```
import mlflow.pyfunc

class AutoResNet150(mlflow.pyfunc.PythonModel):

    def predict_from_picture(self, img_df):
    import cv2 as cv
    import numpy as np
    import base64

    # decoding of base64 encoded image used for transport over http
    img = np.frombuffer(base64.b64decode(img_df[0][0]), dtype=np.uint8)
    img_res = cv.resize(cv.imdecode(img, flags=1), (224, 224), cv.IMREAD_UNCHANGED)
    rgb_img = np.expand_dims(img_res, 0)

    preds = self.keras_model.predict(rgb_img)
    prob = np.max(preds)

    class_id = np.argmax(preds)
    return {“label”: self.class_names[class_id][0][0], “prob”: “{:.4}”.format(prob)}

    def load_context(self, context):
    import scipy.io
    import numpy as np
    import h5py
    import keras
    import cloudpickle
    from keras.models import load_model

    self.results = []
    with open(context.artifacts[“cars_meta”], “rb”) as file:
        # load the car classes file
        cars_meta = scipy.io.loadmat(file)
        self.class_names = cars_meta[‘class_names’]
        self.class_names = np.transpose(self.class_names)

        with open(context.artifacts[“scale_layer”], “rb”) as file:
            self.scale_layer = cloudpickle.load(file)

        with open(context.artifacts[“keras_model”], “rb”) as file:
            f = h5py.File(file.name,’r’)
            self.keras_model = load_model(f, custom_objects={“Scale”: self.scale_
    layer})

        def predict(self, context, model_input):
        return self.predict_from_picture(model_input)
```

다음 단계에서는 py_model을 사용하여 Azure Container Instances 서버에 배포할 것입니다. 이는 MLflow의 Azure ML 통합을 사용하면 됩니다.



Azure Container Instances에서 이미지 분류 모델 배포

이제 머신 러닝 모델을 훈련하였고 클라우드에서 MLflow를 사용하여 작업 영역에 모델을 등록했습니다. 마지막 단계에서는 모델을 Azure Container Instances에 웹 서비스로 배포할 것입니다.

웹 서비스는 이미지입니다(이 경우, Docker 이미지). 여기에는 평가 로직과 모델 자체가 포함되어 있습니다. 우리 사례에서는 사용자 정의 MLflow 모델 표현을 사용하는데, 이를 통해 평가 로직이 REST 클라이언트에서 자동차 이미지를 어떻게 가져오고 응답을 구성하는지 제어할 수 있습니다.

```
# Build an Azure ML Container Image for an MLflow model
azure_image, azure_model = mlflow.azureml.build_image(
                  model_uri="{}/py_model"
                       .format(resnet150_latest_run.info.artifact_uri),
                  image_name="car-resnet150",
                  model_name="car-resnet150",
                  workspace=ws,
                  synchronous=True)

webservice_deployment_config = AciWebservice.deploy_configuration()


# defining the container specs
aci_config = AciWebservice.deploy_configuration(cpu_cores=3.0, memory_gb=12.0)

webservice = Webservice.deploy_from_image(
      image=azure_image,
      workspace=ws,
      name="car-resnet150",
      deployment_config=aci_config,
      overwrite=True)

webservice.wait_for_deployment()
```

Container Instances는 워크플로를 테스트하고 이해하기에 좋은 솔루션입니다. 확장할 수 있는 프로덕션 배포의 경우, Azure Kubernetes Service를 사용해보세요. 자세한 내용은 **배포 방법과 위치**를 참조하세요.

CNN 이미지 분류 시작하기

이 문서와 노트북에서는 전체적 워크플로 훈련을 설정하고 Azure에 프로덕션으로 신경망을 배포하는 데 사용하는 주요 기술을 소개합니다. 링크된 노트북에 나와 있는 연습을 통해 Azure Databricks 환경에서 Keras, Databricks Koalas, MLflow, Azure ML 등의 도구를 사용하여 이를 생성하는 데 필요한 단계를 자세히 알아볼 수 있습니다.

개발자 참고 자료

- **노트북:**
  - Stanford Cars 데이터 세트를 HDF5 파일에 로드
  - Koalas를 이미지 보강에 사용
  - Keras로 CNN 훈련
  - Azure ML에 REST 서비스로 모델 배포
- 영상: Databricks에서 딥 컨볼루션 신경망으로 AI 자동차 분류
- GitHub: EvanEames | Cars
- 슬라이드: 자동차 등급 분류를 위한 컨볼루셔널 신경망 구현
- PDF: Databricks에서 컨볼루션 신경망 구현

8장:

Databricks로 위치 정보 데이터 처리

글: Nima Razavi 및 Michael Johns

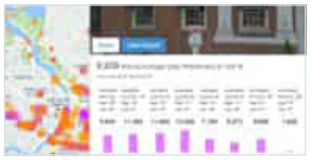
2019년 12월 05일

기술의 진화와 융합으로 시기적절하고 정확한 위치 정보 데이터에 대한 시장이 활성화되었습니다. 매일 수십억 개의 휴대용 기기와 IoT 기기, 수천 개의 무선 및 위성 원격 감지 플랫폼이 수백 엑사바이트의 위치 기반 데이터를 생성합니다. 위치 정보 빅데이터가 폭발적으로 늘어나는 동시에, 머신 러닝의 발전까지 겹쳐 각 산업의 조직들은 새로운 제품과 기능을 개발할 수 있게 되었습니다.

사기 및 악용

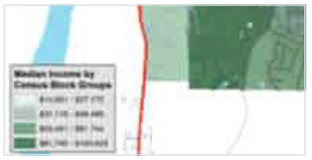
사기 및 충돌 패턴 탐지
(예: 청구 사기, 신용카드 사기)

리테일



부지 선정, 도시 계획,
유동 인구 분석

금융 서비스



경제적 분배, 대출 위험 분석,
리테일 매출 예측, 투자

의료

질병 발생지 확인, 환경이 건강에
미치는 영향, 치료 계획

재해 복구

홍수 조사, 지진 매핑, 대응 계획

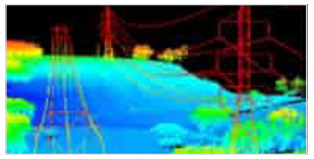
국방 및 정보

정찰, 위협 탐지, 피해 평가

인프라

교통 계획, 농업 관리, 주택 개발

에너지



기후 변화 분석, 에너지 자산 검사,
석유 탐사

위치 정보 데이터를 활용하는 지도는 재해 복구, 국방 및 정보, 인프라, 의료 서비스 등, 여러 사용 사례에 걸쳐 각종 산업에서 널리 활용되고 있습니다.

예를 들어 수많은 기업이 매핑, 부지 검사 등의 지역적 드론 서비스를 제공합니다(**지능적 클라우드 및 지능적 엣지 개발** 참조). 위치 정보 데이터와 관련하여 빠르게 성장하는 산업으로는 자율 주행 자동차도 있습니다. 스타트업이나 기존 기업 모두 차량 센서에서 고도로 컨텍스트화된 위치 정보 데이터 코퍼스를 대량으로 수집하여 자율 주행 자동차 분야에서 차세대 혁신을 이끌고 있습니다(**Databricks, wejo가 모빌리티 데이터 에코시스템 구축 목표를 달성하도록 지원** 참조). 리테일러와 정부 기관에서도 이런 위치 정보 데이터를 활용하고자 합니다. 예를 들어 유동 인구 분석(**유동 인구 인사이트 데이터 세트 구축** 참조)은 새로운 매장을 열기에 최적의 위치를 찾거나, 공공 부문에서는 도시 계획을 개선하는 데 활용할 수 있습니다. 위치 정보 데이터에 대한 투자가 다수 이루어지고 있음에도 불구하고 여러 가지 문제가 존재합니다.

대규모 위치 정보 데이터 분석의 문제

첫 번째 문제는 스트리밍과 배치 애플리케이션으로의 확장입니다. 위치 정보 데이터는 엄청나게 분산되어 있고, 애플리케이션에서 요구하는 SLA도 있어서 기존 스토리지 및 처리 시스템으로는 감당할 수 없습니다. 고객 데이터는 데이터 볼륨, 생성 속도, 스토리지 비용, 엄격한 쓰기 스키마 적용 등, 여러 가지 압력으로 인해 수년 전부터 수직적으로 확장하는 기존 위치 정보 데이터베이스에서 데이터 레이크로까지 흘러넘치고 있습니다. 기업에서는 위치 정보 데이터에 투자하기는 하지만 이런 방대하고 복잡한 데이터 세트를 다운스트림 분석에 사용하도록 준비할 수 있는 적절한 기술 아키텍처를 갖춘 곳은 드뭅니다. 게다가 확장된 데이터는 고급 사용 사례에 필요한 경우가 많기 때문에 AI 중심 이니셔티브 대부분이 시범 사업에서 프로덕션으로 이어지지 못하고 있습니다.

두 번째 문제는 다양한 위치 데이터 형식과의 호환성입니다. 수십 년에 걸쳐 여러 가지 전문화된 **위치 정보 형식**이 생겨난 데다 위치 정보를 얻을 수 있는 부수적 데이터 소스도 많습니다.

- 벡터 형식(예: GeoJSON, KML, Shapefile, WKT)
- 래스터 형식(예: ESRI Grid, GeoTIFF, JPEG 2000, NITF)
- 내비게이션 표준(예: AIS 및 GPS 기기에서 사용하는 형식)
- JDBC / ODBC 연결을 통해 액세스할 수 있는 위치 정보 데이터베이스(예: PostgreSQL / PostGIS)
- Hyperspectral, Multispectral, Lidar 및 Radar 플랫폼의 원격 센서 형식
- OGC 웹 표준(예: WCS, WFS, WMS, WMTS)
- 위치 정보가 태그된 로그, 사진, 영상, 소셜 미디어
- 위치가 참조된 비구조적 데이터

이 블로그 게시물에서는 Databricks Unified Data Analytics Platform을 사용하여 두 가지 주요 문제를 해결하기 위한 일반적 방법을 간단히 알아보겠습니다 이 글은 대량의 위치 정보 데이터를 처리하는 블로그 게시물 시리즈 1부입니다.

Databricks로 위치 정보 워크로드 확장

Databricks는 **빅데이터 분석**과 머신 러닝을 위한 Unified Data Analytics Platform으로, 전 세계적으로 수천 개의 고객사가 사용하고 있습니다. Apache Spark™, Delta Lake, MLflow를 기반으로 하며, 타사 및 사용할 수 있는 라이브러리 통합으로 구성된 광범위한 에코시스템을 구성합니다. **Databricks UDAP**는 엔터프라이즈급 보안, 지원, 안정성 및 성능을 프로덕션 워크로드에서 대규모로 제공합니다. 일반적으로 위치 정보 워크로드는 복잡하고, 모든 사용 사례에 맞는 통합 라이브러리는 존재하지 않습니다. Apache Spark는 기본적으로 위치 정보 데이터 유형을 제공하지는 않지만, 오픈 소스 커뮤니티와 기업에서 공간 라이브러리를 개발하는 데 큰 노력을 기울인 덕분에 선택할 수 있는 옵션이 다양해졌습니다.

일반적으로 공간 조인, 가장 가까운 지역 등, 위치 정보 작업을 확장하기 위한 패턴은 3가지가 있습니다.

1. Apache Spark를 위치 정보 분석으로 확장하는 전용 라이브러리를 사용합니다. 고객들은 GeoSpark, **GeoMesa**, **GeoTrellis**, **Rasterframes**와 같은 라이브러리를 사용합니다. 이러한 프레임워크는 여러 언어 바인딩을 제공하는 경우가 많고 공식화되지 않은 방법보다 확장성과 성능이 우수한 경우가 많지만, 배우는 데 시간이 필요합니다.
2. **GeoPandas**, **Geospatial Data Abstraction Library (GDAL)**, **Java Topology Service (JTS)** 등의 단일 노드 라이브러리를 임시 사용자 정의 함수(UDF)에 래핑해서 Spark DataFrames로 분산 처리합니다. 이 방법은 코드를 그다지 재작성하지 않고도 기존 워크로드를 확장할 수 있는 가장 간단한 방법이지만 전체를 들어서 옮기는(lift-and-shift) 성격이기 때문에 성능이 저하될 수 있습니다.
3. 대량의 워크로드나 컴퓨터 연산이 제한된 워크로드를 처리할 때 그리드 시스템으로 데이터를 색인하고 공간 작업을 실행하는 방법을 사용하는 것이 일반적입니다. 이런 그리드 시스템으로는 **S2**, **GeoHex** , Uber의 **H3** 등이 있습니다. 그리드는 수량이 정해진 식별할 수 있는 셀로 폴리곤, 점 등의 지형을 추정하여 비용이 많이 들어가는 위치 정보 작업을 피하고 더 나은 확장성을 제공합니다. 구현할 때 다소 손실은 있지만 성능은 우수한 단일 정확도에 그리드를 고정할지, 성능은 다소 떨어지지만 손실이 완화되는 다중 정확도에 그리드를 고정할지 결정할 수 있습니다.

다음의 예시는 뉴욕시 택시 승하차 데이터 세트를 다루며, 여기에서 확인할 수 있습니다. 구조가 포함된 **뉴욕 택시 구역** 데이터도 폴리곤 세트로 사용됩니다. 이 데이터에는 뉴욕시 5개 구와 주변 지역에 대한 폴리곤이 포함되어 있습니다. 이 노트북에서는 초기 CSV 파일을 Delta Lake 테이블로 변환하여 신뢰할 수 있고 성능이 적절한 데이터 소스로 사용하기 위한 준비 및 정리 작업을 자세히 설명합니다.

기본 DataFrame은 Databricks를 사용하여 Delta Lake Table에서 읽은 택시 승하차 데이터입니다.

```
%scala
val dfRaw = spark.read.format(“delta”).load(“/ml/blogs/geospatial/delta/nyc-green”)
display(dfRaw) // showing first 10 columns
```

vendor_id	pickup_datetime	dropoff_datetime	store_and_forward	rate_code_id	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passener_count
2	2017-09-30 23:48:04	2017-09-30 23:57:43	N	1	82	7	2	1.89	9
2	2017-09-30 23:50:24	2017-09-30 23:55:30	N	1	25	181	6	1.26	6
2	2017-09-30 23:28:29	2017-09-30 23:37:29	N	1	41	159	1	2.28	9
2	2017-09-30 23:46:44	2017-09-30 23:54:59	N	1	42	41	1	1.09	7
2	2017-09-30	2017-09-30 23:31:49	N	1	33	189	1	2.35	10

Showing the first 1000 rows.

예: Databricks를 사용하여 a Delta Lake 표에서 읽은 위치 공간 데이터

Apache Spark용 위치 정보 라이브러리를 사용한 위치 정보 작업

최근 몇 년 사이에 Apache Spark의 위치 정보 분석 기능을 확장하기 위한 라이브러리가 다수 개발되었습니다. 이 프레임워크는 흔히 적용되는 사용자 정의 유형(UDT)과 함수(UDF)를 일관적인 방식으로 등록하여, 사용자와 개발팀에서 임시 공간 로직을 작성해야 할 부담을 덜어줍니다. 이 블로그 게시물에서는 다양한 기능을 특징적으로 보여주기 위해 여러 가지 공간 프레임워크를 사용합니다. 그 외에도 다른 라이브러리가 존재하며, 이들은 Databricks에서 공간 워크로드를 처리하는 데 사용할 수도 있습니다.

앞서 기본 데이터를 DataFrame에 로드해두었습니다. 이제 경도/위도 속성을 점 구조로 바꾸어야 합니다. 이를 위해서는 UDF를 사용하여 DataFrames에서 작업을 분산 처리해야 합니다. 블로그 끝에 제공되는 노트북에서 이 프레임워크를 클러스터에 추가하고 UDF와 UDT를 등록하기 위한 초기화 호출을 보내는 자세한 방법을 확인하실 수 있습니다. 먼저 GeoMesa(벡터 데이터 처리를 위해 개조한 프레임워크)를 클러스터에 추가했습니다. 데이터 입력의 경우, Spark SQL을 사용하여 JTS를 통합합니다. 그러면 등록된 JTS 기하 클래스로 쉽게 변환해서 사용할 수 있습니다. 위도와 경도를 제공하는 st_makePoint 함수로 점 구조 개체를 생성하겠습니다. 이 함수는 UDF이므로 열에 직접 적용할 수 있습니다.

```
%scala
val df = dfRaw
 .withColumn(“pickup_point”, st_makePoint(col(“pickup_longitude”), col(“pickup_
latitude”)))
 .withColumn(“dropoff_point”, st_makePoint(col(“dropoff_longitude”),col(“dropoff_
latitude”)))
display(df.select(“dropoff_point”,”dropoff_datetime”))
```

(2) Spark Jobs

dropoff_point	dropoff_datetime
POINT (-73.98411560058594 40.695980072021484)	2016-04-01 00:05:53
POINT (-73.8504409790039 40.724143981933594)	2016-04-01 00:05:55

Showing the first 1000 rows.

예: UDF를 사용하여 DataFrames에서 작업을 분산 처리하고 위치 정보 데이터의 위도/경도 속성을 점 구조로 변환합니다.

또한, 공간 조인을 분산 처리합니다. 이 경우에는 GeoMesa에서 제공한 st_contains UDF를 사용하여 승차 지점에 대해 모든 폴리곤의 공간 조인을 생성합니다.

```
%scala
val joinedDF = wktDF.join(df, st_contains($"the_geom", $"pickup_point")
display(joinedDF.select("zone","borough","pickup_point","pickup_datetime"))
```

(2) Spark Jobs

zone	borough	pickup_point	pickup_datetime
Fort Greene	Brooklyn	POINT (-73.98096466064453 40.689029693603516)	2016-06-09 10:35:08
Crown Heights North	Brooklyn	POINT (-73.95674896240234 40.67413330078125)	2016-06-09 10:42:15
Brooklyn Heights	Brooklyn	POINT (-73.9929428100586 40.69749069213867)	2016-06-09 10:47:38
Brooklyn Heights	Brooklyn	POINT (-73.99117279052734 40.6959114074707)	2016-06-09 10:46:09
Williamsburg (South Side)	Brooklyn	POINT (-73.96204376220703 40.70991516113281)	2016-06-09 10:06:12
East Harlem North	Manhattan	POINT (-73.93933868408203 40.80525207519531)	2016-06-09 10:58:19
Steinway	Queens	POINT (-73.9175796508789 40.769954681396484)	2016-06-09 10:45:41
Morningside Heights	Manhattan	POINT (-73.96385192871094 40.80808639526367)	2016-06-09 10:36:34

Showing the first 1000 rows.

예: GeoMesa에서 제공한 st_contains UDF 등을 사용하여 승차 위치에 대해 모든 폴리곤의 공간 조인 생성

단일 노드 라이브러리를 UDF로 래핑

전용 분산 공간 프레임워크를 사용하는 것 외에도 기존 단일 노드 라이브러리를 임시 UDF로 래핑하고 DataFrames에서 위치 정보 작업을 분산 처리할 수 있습니다. 이 패턴은 모든 Spark 언어 바인딩(예: Scala, Java, Python, R, SQL)에 제공되며, 최소한의 코드 변경으로 기존 워크로드를 활용할 수 있는 간단한 방법입니다. 단일 노드 예시를 보여드리기 위해 뉴욕시 구 데이터를 로드하고 **폴리곤 내 점** 작업에 대해 UDF find_borough(…)를 정의한 뒤, geopandas를 사용하여 각 GPS 위치를 구에 할당하겠습니다. **벡터화된 UDF**를 사용하면 성능을 더욱 향상할 수 있습니다.

```
%python
# read the boroughs polygons with geopandas
gdf = gdp.read_file("/dbfs/ml/blogs/geospatial/nyc_boroughs.geojson")

b_gdf = sc.broadcast(gdf) # broadcast the geopandas dataframe to all nodes of the
cluster
def find_borough(latitude,longitude):
  mgdf = b_gdf.value.apply(lambda x: x["boro_name"] if x["geometry"].
intersects(Point(longitude, latitude))
  idx = mgdf.first_valid_index()
  return mgdf.loc[idx] if idx is not None else None

find_borough_udf = udf(find_borough, StringType())
```

이제 UDF를 적용하여 열을 Spark DataFrame에 추가할 수 있습니다. 그러면 구 이름이 각 승차 지점에 할당됩니다.

```
%python
# read the coordinates from delta
df = spark.read.format(“delta”).load(“/ml/blogs/geospatial/delta/nyc-green”)
df_with_boroughs = df.withColumn(“pickup_borough”, find_borough_udf(col(“pickup_
latitude”),col(pickup_longitude)))
display(df_with_boroughs.select(
  “pickup_datetime”,”pickup_latitude”,”pickup_longitude”,”pickup_borough”))
```

▸ (2) Spark Jobs

pickup_datetime	pickup_latitude	pickup_longitude	pickup_borough
2016-04-01 00:06:39	40.718135833740234	-73.95951080322266	Manhattan
2016-04-01 00:06:28	40.86066818237305	-73.88964080810547	Manhattan
2016-04-01 00:07:25	40.73863983154297	-73.88591766357422	Manhattan
2016-04-01 00:09:44	40.69947814941406	-73.92366790771484	Manhattan
2016-04-01 00:16:02	40.691192626953125	-73.9872055053711	Manhattan
2016-04-01 00:14:52	40.761085510253906	-73.92341613769531	Manhattan
2016-04-01 00:11:00	40.686092376708984	-73.97399139404297	Manhattan
2016-04-01 00:17:17	40.79181671142578	-73.944580078125	Manhattan

Showing the first 1000 rows.

예: 단일 노드 예시의 결과(Geopandas를 사용하여 각 GPS 위치를 뉴욕시 구에 할당)

공간 색인을 위한 그리드 시스템

위치 정보 작업은 근본적으로 많은 컴퓨팅 성능을 필요로 합니다. 폴리곤 내 점, 공간 조인, 가장 가까운 지역, 경로 스내핑은 모두 복잡한 작업이 필요합니다. 그리드 시스템으로 색인을 하는 이유는 위치 정보 작업을 모두 배제하기 위해서입니다. 근사치를 산출하지 않고도 가장 확장성이 우수한 구현을 만들 수 있습니다. H3를 사용한 간단한 예시를 보여드리겠습니다.

H3를 사용한 공간 확장 작업은 기본적으로 2단계로 구성됩니다. 1단계는 각 지형(예: 점, 폴리곤)에 대해 H3 색인을 연산하여 UDF geoToH3(…)로 정의하는 것입니다. 2단계는 공간 조인(예: 폴리곤 내 점, 가장 가까운 지역)을 비롯한 공간 작업에 이 색인을 사용하는 것입니다. 이 사례에서는 UDF multiPolygonToH3(…)로 정의합니다.

```
%scala
import com.uber.h3core.H3Core
import com.uber.h3core.util.GeoCoord
import scala.collection.JavaConversions._
import scala.collection.JavaConverters._

object H3 extends Serializable {
  val instance = H3Core.newInstance()
}

val geoToH3 = udf{ (latitude: Double, longitude: Double, resolution: Int) =>
  H3.instance.geoToH3(latitude, longitude, resolution)
}

val polygonToH3 = udf{ (geometry: Geometry, resolution: Int) =>
  var points: List[GeoCoord] = List()
  var holes: List[java.util.List[GeoCoord]] = List()
  if (geometry.getGeometryType == “Polygon”) {
    points = List(
      geometry
        .getCoordinates()
        .toList
        .map(coord => new GeoCoord(coord.y, coord.x)): _*)
  }
  H3.instance.polyfill(points, holes.asJava, resolution).toList
}
```

```
val multiPolygonToH3 = udf{ (geometry: Geometry, resolution: Int) =>
  var points: List[GeoCoord] = List()
  var holes: List[java.util.List[GeoCoord]] = List()
  if (geometry.getGeometryType == “MultiPolygon”) {
    val numGeometries = geometry.getNumGeometries()
    if (numGeometries > 0) {
      points = List(
        geometry
          .getGeometryN(0)
          .getCoordinates()
          .toList
          .map(coord => new GeoCoord(coord.y, coord.x)): _*)
    }
    if (numGeometries > 1) {
      holes = (1 to (numGeometries - 1)).toList.map(n => {
        List(
          geometry
            .getGeometryN(n)
            .getCoordinates()
            .toList
            .map(coord => new GeoCoord(coord.y, coord.x)): _*).asJava
      })
    }
  }
  H3.instance.polyfill(points, holes.asJava, resolution).toList
```

이제 두 개의 UDF를 뉴욕시 택시 데이터와 구 폴리곤 세트에 적용하여 H3 색인을 생성합니다.

```
%scala
val res = 7 //the resolution of the H3 index, 1.2km
val dfH3 = df.withColumn(
  “h3index”,
  geoToH3(col(“pickup_latitude”), col(“pickup_longitude”), lit(res))
)
val wktDFH3 = wktDF
  .withColumn(“h3index”, multiPolygonToH3(col(“the_geom”), lit(res)))
  .withColumn(“h3index”, explode($”h3index”))
```

위도/경도 지점 세트와 폴리곤 구조 세트가 있으면 h3index 필드를 조인 조건으로 사용하여 공간 조인을 실행할 수 있습니다. 이렇게 할당한 후 각 폴리곤 내에 있는 점 개수를 집계할 수 있습니다. 일반적으로 수천, 수만 개의 폴리곤에 매칭해야 하는 점은 수백, 수십억 개에 달합니다. 그래서 확장할 수 있는 방식이 필요합니다. 이 블로그에서 다루지는 않았지만, 근사치로 부족할 때 공간 작업을 지원하기 위한 색인에 사용할 수 있는 다른 기술도 있습니다.

```
%scala
val dfWithBoroughH3 = dfH3.join(wktDFH3,”h3index”)

display(df_with_borough_h3.select(“zone”,”borough”,”pickup_point”,”pickup_
datetime”,”h3index”))
```

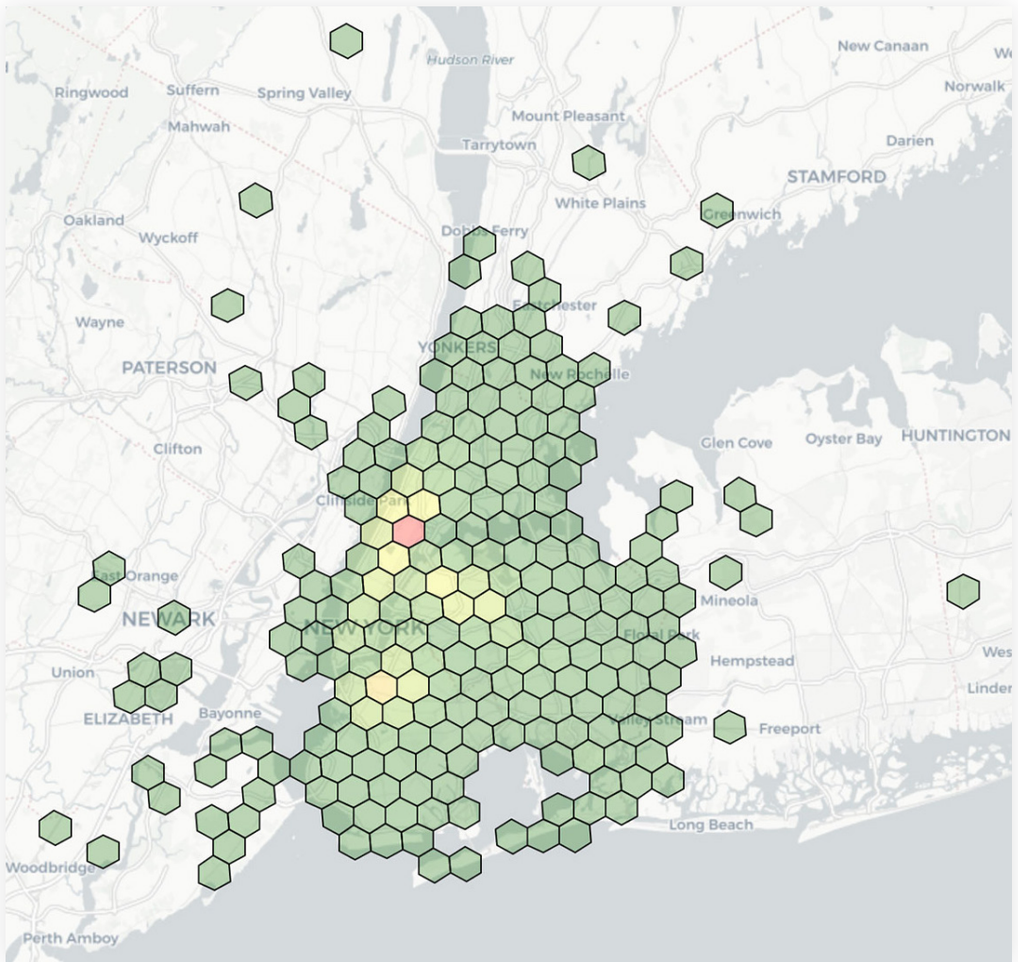
▸ (1) Spark Jobs

zone	borough	pickup_point	pickup_datetime	h3index
Morningside Heights	Manhattan	POINT (-73.95296478271484 40.80758285522461)	2016-06-09 10:14:34	613229523000885247
Central Harlem	Manhattan	POINT (-73.94908905029297 40.80293655395508)	2016-06-09 10:04:08	613229523028148223
Brooklyn Heights	Brooklyn	POINT (-73.99422454833984 40.69488525390625)	2016-06-09 10:52:24	613229551411003391
Van Nest/Morris Park	Bronx	POINT (-73.84475708007812 40.847774505615234)	2016-06-09 10:23:52	613229520937287679
Astoria	Queens	POINT (-73.9139633178711 40.76524353027344)	2016-06-09 10:25:38	613229524726841343
Morningside Heights	Manhattan	POINT (-73.95944213867188 40.80912399291992)	2016-06-09 10:42:56	613229523000885247
Park Slope	Brooklyn	POINT (-73.98164367675781 40.66694641113281)	2016-06-09 10:29:28	613229552660905983
Park Slope	Brooklyn	POINT (-73.97588348388672 40.67397689819336)	2016-06-09 10:53:01	613229552669294591

Showing the first 1000 rows.

예: 위치/경도 지점과 폴리곤 구조 세트의 공간 조인을 나타내는 DataFrame 표, 특정 필드를 조인 조건을 사용

택시 하차 위치를 시각화한 그림은 다음과 같습니다. 위도와 경도가 해상도 7(1.22km 엣지 길이)에 빈으로 표시되어 있고 각 빈 내에 집계된 개수를 기준으로 색을 채웠습니다.



예: 택시 하차 위치를 시각화한 그림. 위도와 경도가 해상도 7(1.22km 엣지 길이)에 빈으로 표시되어 있고 각 빈 내에 집계된 개수를 기준으로 색을 채웠습니다.

Databricks로 공간 형식 처리

위치 정보 데이터에는 위도, 경도와 같이 지표면에서의 물리적 위치나 범위에 대한 기준점과 속성에서 설명하는 지형이 포함되어 있습니다. 선택할 수 있는 파일 형식은 여러 가지가 있지만 여기에서는 몇 가지 대표적 벡터와 래스터 형식을 선택해 Databricks로 판독값을 표현해보겠습니다.

벡터 데이터

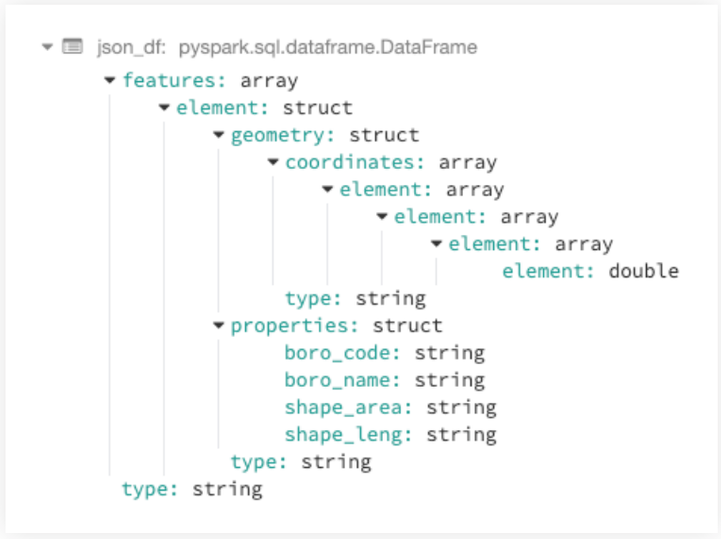
벡터 데이터는 x(경도), y (위도) 좌표를 기준으로 각도로 저장된 세계의 표현입니다. 고도를 고려할 경우 z(m 기준 고도)로도 표현합니다. 벡터 데이터의 세 가지 기본 기호 유형은 점, 선, 폴리곤입니다. Well-known-text (WKT), GeoJSON 및 Shapefile는 벡터 데이터를 저장하는 데 일반적으로 사용하는 형식이며, 아래에 나와 있습니다.

WKT로 저장된 위치 정보로 뉴욕시 택시 구역 데이터를 읽어보겠습니다. 지금 살펴보려는 데이터 구조는 DataFrame입니다. 블로그의 다른 곳에서 사용되었던 것과 마찬가지로 다른 API 및 사용할 수 있는 데이터 소스로도 표준화할 수 있습니다. the_geom 필드에서 발견된 WKT 텍스트는 st_geomFromWKT(…) UDF를 호출하여 해당 JTS 기하 클래스로 쉽게 변환할 수 있습니다.

```
%scala
val wktDFText = sqlContext.read.format(“csv”)
 .option(“header”, “true”)
 .option(“inferSchema”, “true”)
 .load(“/ml/blogs/geospatial/nyc_taxi_zones.wkt.csv”)

val wktDF = wktDFText.withColumn(“the_geom”, st_geomFromWKT(col(“the_geom”))).cache
```

GeoJSON은 지형, 속성, 공간 범위를 포함해 다양한 위치 정보 데이터 구조를 인코딩하기 위해 여러 오픈 소스 GIS 패키지에서 사용됩니다. 이 예시에서는 워크플로에 따라 방법을 변경하여 뉴욕시 구 경계를 읽어보겠습니다. 데이터는 JSON 형식을 따르기 때문에 Databricks 내장 JSON 리더에서 .option(“multiline”,”true”)을 사용하여 중첩된 스키마로 데이터를 로드할 수 있습니다.

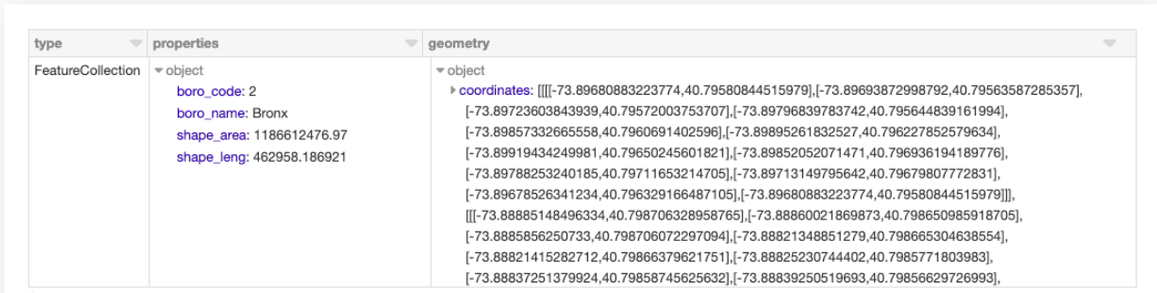
```
%python
json_df = spark.read.option(“multiline”,”true”).json(“nyc_boroughs.geojson”)
```



예: Databricks 내장 JSON 리더에서 .option(“multiline”,”true”)을 사용하여 중첩된 스키마로 데이터 로드

여기에서 Spark에 내장된 explode 함수를 사용하여 최상위 수준 열까지 모든 필드를 올릴 수도 있습니다. 예를 들어 기하 구조, 속성, 타입을 가져와서 기하 구조를 해당 JTS 클래스로 변환할 수 있습니다. 이는 WKT 예시에서 확인할 수 있습니다.

```
%python
from pyspark.sql import functions as F
json_explode_df = ( json_df.select(
 “features”,
 “type”,
 F.explode(F.col(“features.properties”)).alias(“properties”)
).select(“*”,F.explode(F.col(“features.geometry”)).alias(“geometry”)).drop(“features”))

display(json_explode_df)
```

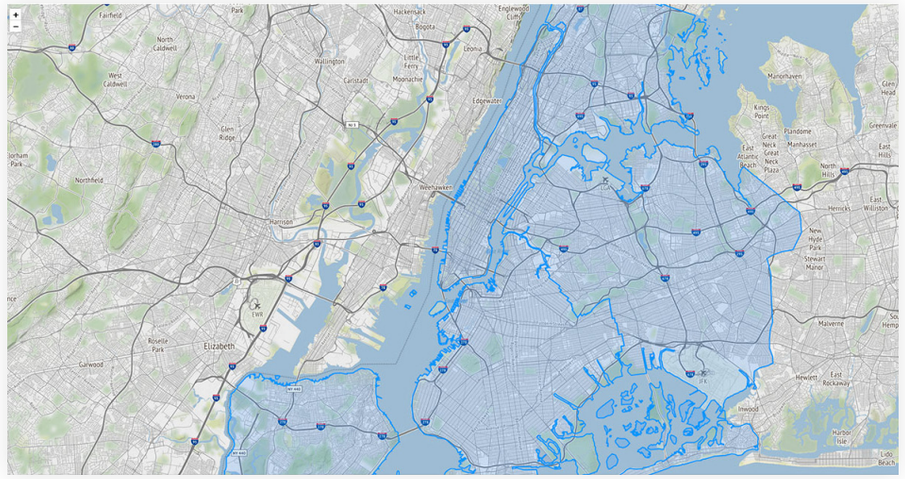


type	properties	geometry
FeatureCollection	object boro_code: 2 boro_name: Bronx shape_area: 1186612476.97 shape_leng: 462958.186921	object coordinates: [[[[-73.89680883223774,40.79580844515979],[-73.89693872998792,40.79563587285357], [-73.89723603843939,40.7957200375370 7],[-73.89796839783742,40.795644839161994], [-73.89857332665558,40.7960691402596],[-73.89895261832527,40.796227852579634], [-73.89919434249981,40.7965024560 1821],[-73.89852052071471,40.796936194189776], [-73.89788253240185,40.79711653214705],[-73.89713149795642,40.79679807772831], [-73.89678526341234,40.79632916648 7105],[-73.89680883223774,40.79580844515979]]], [[[-73.88885148496334,40.798706328958765],[-73.8886002 1869873,40.798650985918705], [-73.8885856250733,40.798706072297094],[-73.88821348851279,40.798665304638554], [-73.88821415282712,40.7986637962 1751],[-73.88825230744402,40.798577180 3983], [-73.88837251379924,40.7985874562 5632],[-73.88839250519693,40.7985662972 6993],

예: Spark의 내장 explode 함수를 사용하여 최상위 수준까지 필드 상승, DataFrame 표에 표시

또한, 노트북에서 기존 DataFrame을 사용하거나 Folium(공간 데이터를 렌더링하는 Python 라이브러리) 등의 라이브러리로 직접 데이터를 렌더링하여 뉴욕시 택시 구역 데이터를 시각화할 수도 있습니다. Databricks File System (DBFS)은 분산된 스토리지 계층 위에 실행되며, 친숙한 파일 시스템 표준을 사용하여 데이터 형식으로 작업하도록 코딩할 수 있습니다. DBFS는 FUSE Mount가 있어서 파일 읽기 및 쓰기 작업을 수행하는 로컬 API 호출이 가능하므로 분산되지 않은 API로 데이터를 쉽게 로드하여 인터랙티브 렌더링을 지원할 수 있습니다. 아래의 Python open(…) 명령에서 "/dbfs/…" 접두사가 있으면 FUSE Mount를 사용할 수 있습니다.

```
%python
import folium
import json

with open ("/dbfs/ml/blogs/geospatial/nyc_boroughs.geojson", "r") as myfile:
 boro_data=myfile.read() # read GeoJSON from DBFS using FuseMount

m = folium.Map(
 location=[40.7128, -74.0060],
 tiles='Stamen Terrain',
 zoom_start=12
)
folium.GeoJson(json.loads(boro_data)).add_to(m)
m # to display, also could use displayHTML(...) variants
```



예: 노트북에서 기존 DataFrame을 사용하거나 Folium(공간 데이터를 렌더링하는 Python 라이브러리) 등의 라이브러리로 직접 위치 정보 데이터를 렌더링하여 뉴욕시 택시 구역 데이터를 시각화

Shapefile은 ESRI 에서 개발한 인기 있는 벡터 형식으로, 지형의 기하 위치와 속성 정보를 저장합니다. 이 형식은 동일한 디렉터리에 저장되고 공통적인 파일명 접두사(*.shp, *.shx and *.dbf가 필수)가 있는 파일 모음으로 구성됩니다. Shapefile은 KML로 대체할 수도 있는데, 우리 고객사에서도 사용하지만 간단하게 내용을 전달하기 위해 여기에서는 설명을 생략했습니다. 이 예시에서는 NYC Building shapefile을 사용하겠습니다. Shapefile을 읽는 방법은 여러 가지가 있지만 GeoSpark를 사용한 예시를 보여드리겠습니다.

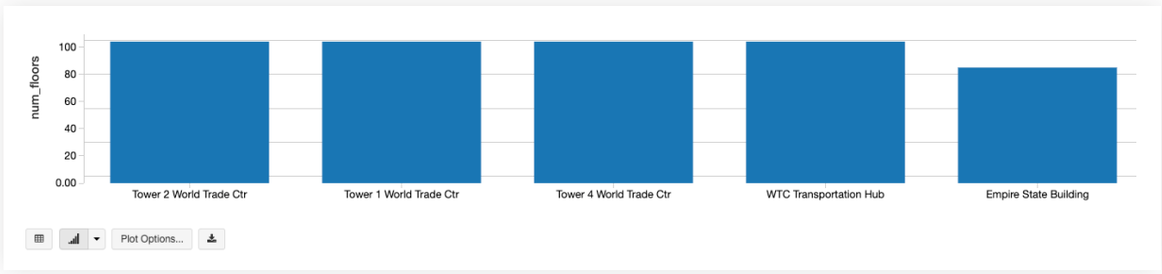
```
%scala
var spatialRDD = new SpatialRDD[Geometry]
spatialRDD = ShapefileReader.readToGeometryRDD(sc, "/ml/blogs/geospatial/shapefiles/nyc")

var rawSpatialDf = Adapter.toDf(spatialRDD,spark)
rawSpatialDf.createOrReplaceTempView("rawSpatialDf") //DataFrame now available to SQL,
Python, and R


display(json_explode_df)
```

기본rawSpatialDf를 임시 뷰로 저장하면 순수 Spark SQL 구문에 들어가 DataFrame으로 작업할 수 있습니다. 예를 들어 UDF를 적용하여 shapefile WKT를 기하 구조로 변환하는 작업도 포함됩니다.

```
%sql
SELECT *,
 ST_GeomFromWKT(geometry) AS geometry -- GeoSpark UDF to convert WKT to Geometry
FROM rawspatialdf
```

또한 Databricks에 내장된 인라인 분석용 시각화 도구를 사용할 수도 있습니다. 뉴욕시에서 가장 높은 건물을 도표로 그리는 작업이 가능합니다.

```
%%sql
SELECT name,
 round(Cast(num_floors AS DOUBLE), 0) AS num_floors --String to Number
FROM rawspatialdf
WHERE name <> ‘’
ORDER BY num_floors DESC LIMIT 5
```



예: 인라인 분석 그래프 작성을 위한 Databricks의 기본 시각화(예: 뉴욕시에서 가장 높은 건물)

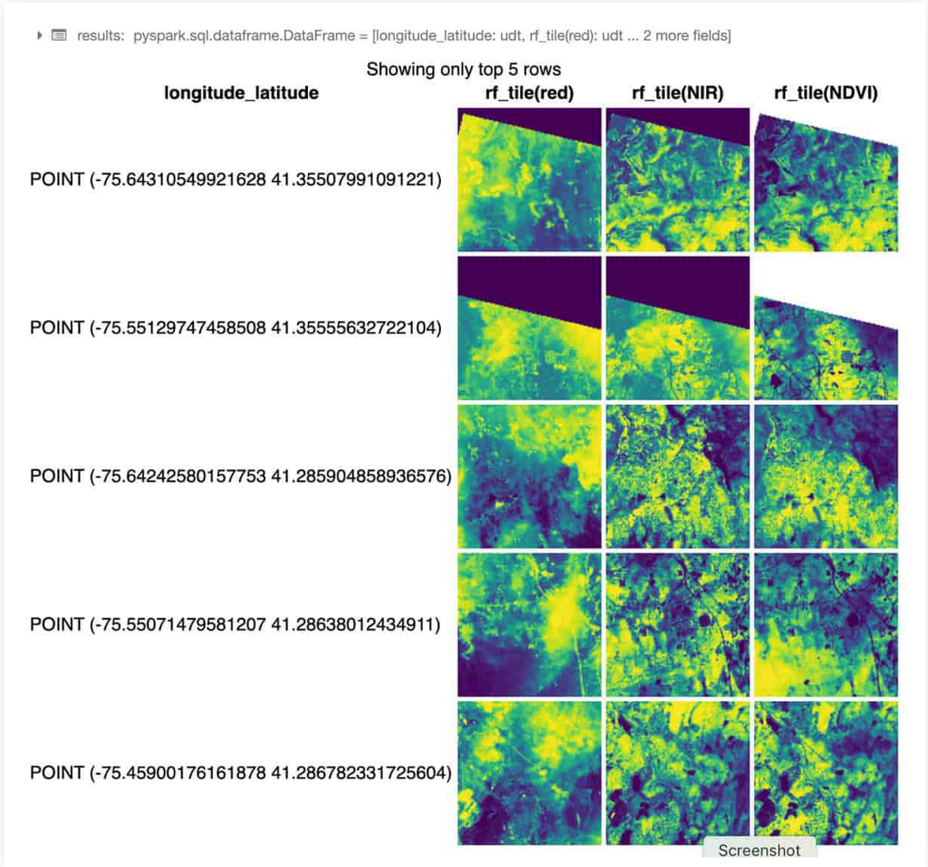
래스터 데이터

래스터 데이터는 (이산적, 또는 연속적인) 행과 열로 구성된 셀(또는 픽셀) 행렬로 지형 정보를 저장합니다. 위성 이미지, 사진 측량, 스캔된 지도는 모두 래스터 기반 지상 관측(EO) 데이터 유형입니다.

다음의 Python 예시는 RasterFrames(DataFrame 중심적 공간 분석 프레임워크)를 사용하여 GeoTIFF Landsat-8 이미지(적외선, 근적외선)의 **두 대역**을 읽고 **정규 식생 지수**로 결합합니다 이 데이터를 사용해서 뉴욕시 주변의 식물 건강을 평가할 수 있습니다. rf_ipython 모듈은 RasterFrame 콘텐츠를 다양한 시각적으로 유용한 형식으로 변환하는 데 사용할 수 있습니다. 예를 들어 아래에서 빨간색 NIR 및 NDVI 타일 열은 Databricks 기본 displayHTML(…) 명령을 사용하여 색 램프로 렌더링되고 노트북 내에 결과를 표시합니다.

```
%python
# construct a CSV “catalog” for RasterFrames `raster` reader
# catalogs can also be Spark or Pandas DataFrames
bands = [f’B{b}’ for b in [4, 5]]
uris = [f’https://landsat-pds.s3.us-west-2.amazonaws.com/c1/L8/014/032/LC08_L1TP_0140
32_20190720_20190731_01_T1/LC08_L1TP_014032_20190720_20190731_01_T1_{b}.TIF’ for b in
bands]
catalog = ‘,’.join(bands) + ‘\n’ + ‘,’.join(uris)

# read red and NIR bands from Landsat 8 dataset over NYC
rf = spark.read.raster(catalog, bands) \
 .withColumnRenamed(‘B4’, ‘red’).withColumnRenamed(‘B5’, ‘NIR’) \
 .withColumn(‘longitude_latitude’, st_reproject(st_centroid(rf_geometry(‘red’)), rf_
crs(‘red’), lit(‘EPSG:4326’))) \
 .withColumn(‘NDVI’, rf_normalized_difference(‘NIR’, ‘red’)) \
 .where(rf_tile_sum(‘NDVI’) > 10000)

results = rf.select(‘longitude_latitude’, rf_tile(‘red’), rf_tile(‘NIR’), rf_
tile(‘NDVI’))
displayHTML(rf_ipython.spark_df_to_html(results))
```

RasterFrames는 사용자 정의 **Spark DataSource**를 통해 GeoTIFF, JP2000, MRF, HDF 등의 여러 가지 래스터 형식을 **다양한 서비스**에서 읽을 수 있습니다. 또한, 벡터 형식 GeoJSON과 WKT/WKB도 지원합니다. RasterFrame 콘텐츠는 **200개 이상의 래스터 및 벡터 함수**를 통해 필터링, 변환, 요약, 리샘플, 래스터화가 가능합니다. 위의 예시에서는 st_reproject(…)와 st_centroid(…)를 사용했습니다. 이는 Python, SQL, Scala용 API를 제공하고 Spark ML과의 상호운용성을 제공합니다.



> **예:** RasterFrame 콘텐츠는 200개 이상의 래스터 및 벡터 함수를 통해 필터링, 변환, 요약, 리샘플링, 래스터화가 가능

위치 정보 데이터베이스

위치 정보 데이터베이스는 소규모 데이터는 파일로 제공하고, 중간 규모 데이터는 JDBC / ODBC 연결을 통해 액세스할 수 있습니다. Databricks를 사용하여 기본 JDBC / ODBC 데이터 소스로 여러 SQL 데이터베이스를 쿼리할 수 있습니다. PostgreSQL에 연결하는 방법은 아래와 같습니다. 이는 일반적으로 소규모 워크로드에서 PostGIS 확장 프로그램을 적용하여 사용합니다. 이 연결 패턴을 사용하면 기존 데이터베이스에 대한 액세스를 그대로 유지할 수 있습니다.

```
%scala
display(
  sqlContext.read.format(“jdbc”)
  .option(“url”, jdbcUrl)
  .option(“driver”, “org.postgresql.Driver”)
  .option(“dbtable”,
    “””(SELECT * FROM yellow_tripdata_staging
    OFFSET 5 LIMIT 10) AS t”””) //predicate pushdown
  .option(“user”, jdbcUsername)
  .option(“jdbcPassword”, jdbcPassword)
.load)
```

vendor_id	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	rate_code_id	store_and_fwd_flag	pickup_location_id	dropoff_location_id	payment_type	fare_amount
2	2019-01-06 16:27:40	2019-01-06 16:29:47	5	.16	1	N	142	142	4	-3
2	2019-01-06 16:27:40	2019-01-06 16:29:47	5	.16	1	N	142	142	2	3
2	2019-01-06 16:51:27	2019-01-06 17:05:55	5	1.99	1	N	239	230	2	11
1	2019-01-06 16:38:49	2019-01-06 16:58:05	1	2.10	1	N	164	163	2	13
1	2019-01-06 16:59:54	2019-01-06 17:09:33	4	1.40	1	N	163	186	2	8
2	2019-01-06 16:25:58	2019-01-06 16:35:36	3	1.77	1	N	137	90	1	8.5
2	2019-01-06 16:42:45	2019-01-06 16:47:05	3	.67	1	N	68	234	2	5
2	2019-01-06 16:50:21	2019-01-06 16:57:03	2	1.09	1	N	234	100	1	6.5

Databricks에서 위치 정보 분석 시작하기

기업과 정부 기관에서는 기업 데이터 소스와 더불어 공간이 참조된 데이터를 사용하여 실천할 수 있는 인사이트를 얻고 다양한 혁신적 사용 사례를 제공합니다. 이 블로그에서는 Databricks Unified Data Analytics Platform을 사용하면 얼마나 쉽게 위치 정보 워크로드를 확장하고, 클라우드에서 방대한 데이터를 캡처, 저장, 분석하도록 지원할 수 있는지 알아보았습니다.

다음 블로그에서는 Databricks를 사용한 대규모 위치 정보 처리에 대한 고급 주제를 자세히 살펴보겠습니다. 데이터 준비 노트북, GeoMesa + H3 노트북, GeoSpark 노트북, GeoPandas 노트북, Rasterframes 노트북에서 공간 형식과 특징적 프레임워크에 대한 자세한 정보를 확인하실 수 있습니다. 또한, 위치 정보 주제와 관련된 문서 의 새로운 섹션도 수시로 확인해주세요.

9장: 고객 사용 사례

Comcast는 엔터테인먼트의 미래를 제공합니다

"Databricks를 사용하면 더욱 정보에 입각한 결정을, 더욱 빠르게 내릴 수 있습니다."

— **Jim Forsythe**
선임 이사, Comcast 제품 분석 및 행동 과학 부문

Comcast는 수백만 명의 고객에게 개인화된 경험을 연결해주는 글로벌 기술 미디어 기업이지만, 어마어마하게 많은 데이터, 취약한 데이터 파이프라인, 데이터 사이언스 협업 부족으로 인해 어려움을 겪었습니다. Delta Lake, MLflow와 함께 Databricks를 사용한 덕분에 페타바이트 규모의 데이터에 적절한 성능을 제공하는 데이터 파이프라인을 구축하였고, 수백 개의 모델 수명 주기를 간편하게 관리함으로써 음성 인식과 머신 러닝을 활용하는 혁신적이고 독창적인 시청자 환경을 구현하였고 수상도 했습니다.

사용 사례: 경쟁이 치열한 엔터테인먼트 산업에는 '일시 정지' 버튼을 누를 여유는 없습니다. Comcast는 데이터 입력에서 고객이 만족할 만한 새로운 기능을 제공하는 머신 러닝 모델 배포에 이르기까지 모든 기술을 현대화해야 할 필요성을 느꼈습니다.

솔루션 및 이점: Comcast는 통합 분석 전략을 도입한 덕분에 미래형 AI 기반 엔터테인먼트로 나아갈 수 있었습니다. 독보적인 고객 환경으로 시청자들이 즐겁게 몰입할 수 있도록 지원하고 있습니다.

- **EMMY상을 수상한 시청자 환경**: Databricks는 Comcast가 시청자 참여를 높여주는 지능적 음성 명령으로 매우 혁신적이고 수상 경력에 빛나는 시청자 환경을 구현하도록 도왔습니다.
- **컴퓨팅 비용 10배 절감**: Delta Lake로 데이터 수집을 최적화하고 나서 640대의 서버를 64대로 줄이면서도 성능까지 개선했습니다. 개발팀은 인프라 관리에 소비하는 시간을 줄이고 분석에 더욱 집중할 수 있습니다.
- **데이터 사이언스 생산성 향상**: Delta Lake로 업그레이드하여 사용한 덕분에 하나의 인터랙티브 업무 공간에서 여러 프로그래밍 언어가 지원되어서 데이터 사이언티스트들의 전반적 협업이 강화되었습니다. 또한, 데이터팀은 데이터 파이프라인에서 언제든 데이터를 사용하고, 새로운 모델을 더욱 빠르게 구축하고 훈련할 수 있게 되었습니다.
- **모델 배포 시간 단축**: Comcast는 현대화를 통해 운영팀이 서로 다른 플랫폼에 모델을 배포하는 시간을 몇 주에서 몇 분으로 단축했습니다.

자세히 알아보기

9장: 고객 사용 사례

Regeneron은 유전체 서열을 포함한 약물 발견을 가속화합니다

REGENERON

"Databricks Unified Data Analytics Platform은 의사 겸 연구자, 컴퓨팅 생물학자에 이르기까지 통합 약물 개발 과정에 참여하는 모든 사람이 모든 데이터에서 인사이트에 쉽게 액세스하고, 분석, 추출할 수 있도록 지원합니다."

— **Jeffrey Reid, Ph.D.**
Regeneron 유전체 정보 책임자

Regeneron는 유전체 데이터를 활용하여 도움이 필요한 환자에게 새로운 약을 제공하는 것을 목표로 합니다. 하지만 이 데이터에서 사람들의 인생을 바꿀 만한 발견과 표적화된 치료법을 얻어내는 것이 그 어느 때보다도 어려워졌습니다. 데이터팀에서는 처리 성능이 부족하고 확장에 한계가 있어서 페타바이트 규모의 유전자 데이터와 임상 데이터를 분석할 수 없었습니다. Databricks는 모든 유전체 데이터 세트를 신속하게 분석하여 새로운 치료법을 발견하는 기간을 단축할 수 있도록 지원합니다.

사용 사례: 현재 약물 개발 파이프라인에 있는 모든 실험적 약물의 95% 이상이 실패할 것으로 예상됩니다. Regeneron Genetics Center는 이런 상황을 개선하기 위해 40만 명 이상의 유전자 시퀀스와 전자 건강 기록을 결합해 세계에서 가장 종합적인 유전체 데이터베이스를 구축했습니다. 그러나 이 방대한 데이터 세트를 분석하는 데는 여러 가지 어려움이 따랐습니다.

- 유전체와 임상 데이터는 매우 분산되어 있어서 전체 10TB 데이터 세트에 대해 모델을 분석하고 훈련하기가 매우 어렵습니다.
- 기존 아키텍처가 800억 개의 데이터 포인트를 분석하도록 지원하기는 어려울 뿐만 아니라 비용도 많이 들어갑니다.
- 데이터팀은 분석에 사용할 수 있도록 데이터에 ETL을 적용하는 데만 며칠이 걸립니다.

솔루션 및 이점: Databricks는 Amazon Web Services에서 실행되는 Unified Data Analytics Platform을 제공하여 운영을 단순화하고 데이터 사이언스의 생산성을 높여 약물 발견 기간을 단축했습니다. Regeneron에서는 이전에는 사용할 수 없던 새로운 방식으로 데이터를 분석할 수 있게 되었습니다.

- **약물 타겟 식별 가속화**: 데이터 사이언티스트와 컴퓨팅 생물학자가 전체 데이터 세트에 쿼리를 실행하는 데 걸리는 시간이 30분에서 3초로 줄어, 600배나 개선되었습니다!
- **생산성 향상**: 협업이 개선되고 DevOps가 자동화되었으며, 파이프라인이 가속화된 덕분에(ETL을 3주에서 2일로 단축) 다양한 연구를 지원할 수 있게 되었습니다.

자세히 알아보기

databricks

9장: 고객 사용 사례

Nationwide는 보험 계리 모델링으로 보험을 혁신합니다

Nationwide

"Databricks를 사용한 이후로 모든 데이터에 대해 더욱 빠르게 모델을 훈련할 수 있어서 정확한 보험료 예측이 가능했고 수익에도 실질적 영향을 미쳤습니다."

— **Bryn Clark**
Nationwide 데이터 사이언티스트

가용한 데이터가 폭발적으로 늘어나고 시장에서 경쟁이 격화되면서 보험사에서도 고객에게 저렴한 보험료를 제공하기 어려워졌습니다. Nationwide는 다운스트림 ML에서 분석해야 할 보험 기록이 수억 건에 달하자, 기존 배치 분석 프로세스가 너무 느리고 부정확해서 청구 빈도와 심각도를 예측하기 위한 인사이트를 얻는 데 한계가 있다는 것을 깨달았습니다. Databricks를 사용하고 나서 대규모로 딥러닝 모델을 적용해 더욱 정확한 보험료를 예측하였고 보험금 청구에서 수익이 늘어났습니다.

사용 사례: 정확한 보험료를 제공하려면 보험 청구 정보를 활용하는 것이 핵심입니다. 하지만 자주 발생하지 않고 예측 불가능한 청구의 성격상, 가변적인 보험 기록을 분석할 때의 데이터 문제를 해결하기란 만만치 않았고 부정확한 보험료가 산출되었습니다.

솔루션 및 이점: Nationwide는 Databricks Unified Data Analytics Platform을 사용하여 데이터 입력에서 딥러닝 모델 배포에 이르기까지 모든 분석 프로세스를 관리합니다. 이 완전 관리형 플랫폼으로 IT 운영을 단순화하고 데이터 사이언스팀에 새로운 데이터 중심적 기회를 제공했습니다.

- **대규모 데이터 처리**: 전체 데이터 파이프라인의 런타임을 34시간에서 4시간으로 단축해, 성능을 9배 향상했습니다.
- **Featurization 시간 단축**: 데이터 엔지니어링을 통해 5시간 걸리던 작업을 약 20분으로 줄여서 15배 빠르게 feature를 찾아낼 수 있게 되었습니다.
- **빠른 모델 훈련**: 훈련 시간을 50% 단축해서 새로운 모델의 시장 출시 시간을 줄였습니다.
- **모델 평가 개선**: 모델 평가를 3시간에서 5분 미만으로 줄여서 60배 개선했습니다.

9장: 고객 사용 사례

Condé Nast는 데이터와 AI 기반 경험으로 독자 참여를 끌어올립니다

CONDÉ NAST

"Databricks는 믿기 힘들 정도로 강력한 엔드투엔드 솔루션이 되어주었습니다. 전문성이 서로 다른, 다양한 팀원이 신속히 모여서 방대한 데이터를 활용해 실천할 수 있는 비즈니스 결정을 내릴 수 있게 되었습니다."

— **Paul Fryzel**
Condé Nast의 AI 인프라 수석 엔지니어

Condé Nast는 The New Yorker, Wired, Vogue 등 세계적으로 유명한 잡지를 보유하고 있는 세계 유수의 미디어 기업 중 하나입니다. Condé Nast는 데이터를 사용해서 인쇄물, 온라인, 동영상, 소셜 미디어를 통해 1억 명 이상과 연결합니다.

사용 사례: 주요 언론사인 Condé Nast는 포트폴리오에서 20개 이상 브랜드를 관리합니다. 매월 웹 사이트는 1억 명 이상 방문하고, 페이지 조회수는 8억 회 이상에 달해서 엄청난 양의 데이터가 발생합니다. 데이터팀은 머신 러닝을 사용하여 사용자 참여를 개선하고, 개인화된 콘텐츠 추천과 표적화된 광고를 제공하는 데 집중합니다.

솔루션 및 이점: Databricks는 Condé Nast에 운영을 단순화하고, 우수한 성능을 제공하면서도 데이터 사이언스 혁신을 지원하는 완전 관리형 클라우드 플랫폼을 제공합니다.

- **고객 참여 개선**: 개선된 데이터 파이프라인을 통해, Condé Nas는 더욱 빠르고 정확하고 유익한 콘텐츠를 추천 할 수 있었고, 이를 통해 사용자 환경도 개선되었습니다.
- **확장할 수 있는 설계**: 규모와 관계없이 데이터 세트를 처리하고 인사이트를 얻을 수 있습니다.
- **모델의 프로덕션 배포 증가**: MLflow를 사용한 이후로 데이터 사이언스팀은 제품을 더욱 빠르게 혁신할 수 있습니다. 1,200개 이상의 모델을 프로덕션에 배포했습니다.

자세히 알아보기

databricks

9장: 고객 사용 사례

Showtime은 ML을 이용한 데이터 기반 콘텐츠 프로그래밍을 제공합니다

"Databricks 플랫폼을 사용한 덕분에 데이터 사이언티스트만으로 구성된 전담팀이 그동안 문제가 되었던 구성 문제를 모두 뒤로 하고 엄청난 도약을 할 수 있었습니다. 생산성이 극적으로 개선되었습니다."

— **Josh McNutt**
SHOWTIME 데이터 전략 및 소비자 분석 전무

SHOWTIME®은 프리미엄 TV 네트워크 및 스트리밍 서비스이며, "Shameless," "Homeland," "Billions," "The Chi," "Ray Donovan," "SMILF," "The Affair," "Patrick Melrose," "Our Cartoon President," "Twin Peaks" 등과 같은 수상 경력에 빛나는 오리지널 시리즈와 오리지널 리미티드 시리즈를 제공합니다.

사용 사례: SHOWTIME의 데이터 사이언스팀은 조직 전체에서 데이터와 분석을 민주화하는 데 초점을 맞춥니다. SHOWTIME은 방대한 구독자 데이터(예: 시청한 방송, 시간대, 사용한 기기, 구독 기록)를 수집하여 머신 러닝으로 구독자의 행동을 예측하고, 방송 편성과 프로그래밍을 개선합니다.

솔루션 및 이점: Databricks는 SHOWTIME이 조직 전반에 데이터와 머신 러닝을 민주화하고 더욱 데이터 중심적 문화를 조성하도록 도왔습니다.

- **파이프라인 6배 가속**: 24시간 이상 걸리던 데이터 파이프라인이 4시간 미만으로 실행되어, 팀에서 더욱 빠르게 결정을 내릴 수 있습니다.
- **인프라 복잡성 제거**: 클라우드에서 자동 클러스터 관리를 활용하는 완전 관리형 플랫폼을 사용한 데이터 사이언스팀은 하드웨어 구성, 클러스터 프로비저닝, 디버깅 등을 대신해 머신 러닝에 집중할 수 있습니다.
- **구독자 환경 혁신**: 데이터 사이언스에서의 협업과 생산성이 개선되면서 새로운 모델과 기능을 출시하는 시간이 단축되었습니다. 더욱 신속하게 실험하고, 구독자에게 더욱 개인화된 우수한 환경을 제공할 수 있습니다.

자세히 알아보기

databricks

9장: 고객 사용 사례

Shell은 더 깨끗한 세상을 위한 에너지 솔루션으로 혁신합니다

"Databricks는 우리 Shell에게 엄청난 가치를 제공해주었습니다. [Databricks 기반의] 재고 최적화 도구는 우리 조직에서 나온 최초의 확장된 디지털 제품으로, 이제 전 세계에 배포되어서 매년 수백만 달러를 절약할 수 있게 되었습니다."

— **Daniel Jeavons**
Shell 고급 분석 부문 혁신 센터 대표

Shell은 석유 가스 탐사 및 생산 기술로 유명한 선도적 기업으로서, 석유, 천연가스 생산, 휘발유 및 천연가스 판매, 석유화학 제품 제조 분야에서 세계 최고 수준을 자랑합니다.

사용 사례: Shell은 생산을 유지하기 위해 전 세계 시설에 3,000개 이상의 예비 부품을 보관합니다. 적절한 시점에 적절한 부품을 제공해야 생산이 중단되지 않지만, 비용을 늘리는 과도한 재고 축적을 방지하는 것도 그만큼 중요합니다.

솔루션 및 이점: Databricks는 Shell에 재고 및 공급망 관리를 개선하는 데 도움이 되는 클라우드 기반 통합 분석 플랫폼을 제공합니다.

- **예측 모델링**: 확장할 수 있는 예측 모델은 50개 이상의 위치에서 3,000개 이상의 재료 유형에 대해 개발, 배포됩니다.
- **과거 데이터 분석**: 각 재료 모델별로 Markov Chain Monte Carlo를 1만 회 반복 시뮬레이션하여 과거의 문제 발생 분포를 찾아냅니다.
- **엄청난 성능 향상**: 데이터 사이언스팀은 성능 향상에 집중하여 Databricks의 50노드 Apache Spark™ 클러스터에서 재고 분석과 예측 시간을 48시간에서 45분으로 단축해, 32배나 성능을 높였습니다.
- **지출 경감**: 연간 수백만 달러에 달하는 비용이 절감됩니다.

자세히 알아보기

9장: 고객 사용 사례

Riot Games는 AI를 이용해 게이머의 참여를 유도하고 이탈을 줄입니다

“데이터 사이언티스트들을 클러스터 관리에서 해방시켜주고 싶었습니다. 사용하기 편리한 관리형 Spark 솔루션을 Databricks에서 사용한 후로 가능하게 되었죠. 이제 우리 팀은 게임 환경을 개선하는 데 집중할 수 있습니다.”

— **Colin Borys**
Riot Games 데이터 사이언티스트

Riot Games의 목표는 세계에서 가장 플레이어 중심적인 게임 기업이 되는 것입니다. 2006년에 로스앤젤레스에서 설립된 Riot Games는 ‘리그 오브 레전드’ 게임으로 가장 유명합니다. 매월 게임을 플레이하는 게이머가 1억 명 이상입니다.

사용 사례: 네트워크 성능 모니터링을 통해 게임 환경을 개선하고 게임 내 욕설을 차단합니다.

솔루션 및 이점: Databricks는 확장 가능하고 빠른 분석을 제공하여 Riot Games가 플레이어의 게임 환경을 개선하도록 도왔습니다.

- **게임 내 구매 환경 개선**: 5,000억 개 이상의 데이터 포인트를 기반으로 고유한 서비스를 제공하는 추천 엔진을 신속히 구축하고 제품화했습니다. 이제 게이머들이 원하는 콘텐츠를 더욱 쉽게 찾을 수 있습니다.
- **게임 지연 완화**: 네트워크 문제를 실시간으로 탐지하는 ML 모델을 구축하였고 플레이어에게 부정적인 영향을 미치기 전에 장애를 해결할 수 있습니다.
- **빠른 분석**: 데이터 준비와 탐색 처리 성능이 EMR에 비해 50% 향상되어서 분석 속도가 상당히 빨라졌습니다.

자세히 알아보기

databricks

9장: 고객 사용 사례

Eneco는 ML을 사용해 에너지 사용량과 운영 비용을 절감합니다

Eneco

"Databricks는 Delta Lake와 Structured Streaming을 활용하여 매우 빠르게 고객에게 알림과 추천을 제공할 수 있도록 도와주었습니다. 고객들은 가정에서 불편해지기 전에 미리 문제를 해결하고 수정할 수 있습니다."

— **Stephen Galsworthy**
Quby 데이터 사이언스 책임자

Quby는 에너지 사용량, 쾌적도, 가정 보안 등을 제어하는 스마트 에너지 관리 기기인 Toon을 개발한 기술 기업입니다. Quby의 스마트 기기는 유럽 전역의 가정에서 사용됩니다. 따라서 가정에서 사용하는 가전의 센서에서 수집한 페타바이트 규모의 IoT 데이터로 구성된 유럽 최대 규모의 에너지 데이터 세트를 관리합니다. Quby는 이 데이터를 사용하여 고객이 더욱 편안한 생활을 영위하면서도 개인 맞춤형 에너지 사용량 추천을 통해 에너지 소비를 절약하도록 지원하고자 합니다.

사용 사례: 개인 맞춤형 에너지 사용량 추천 머신 러닝과 IoT 데이터를 기반으로 Waste Checker 앱에서 가정 내 에너지 사용량을 줄이기 위한 개인 맞춤형 추천을 제공합니다.

솔루션 및 이점: Databricks는 Quby에게 Unified Data Analytics Platform을 제공하여 데이터 사이언스와 엔지니어링에서 확장 가능하고 협력적인 환경을 조성함으로써, 데이터팀이 더욱 빠르게 혁신하고 ML 기반 서비스를 Quby의 고객에게 제공할 수 있도록 했습니다.

- **비용 절감**: Databricks가 제공하는 비용 절감 기능(예: 자동 확장 클러스터, Spot 인스턴스) 덕분에 Quby는 인프라 관리 운영 비용을 상당히 절감하면서도 대량의 데이터를 처리할 수 있게 되었습니다.
- **빠른 혁신**: 기존 아키텍처로는 개념 증명에서 프로덕션까지 12개월 이상 걸렸습니다. Databricks를 사용한 이후로는 같은 프로세스가 8주 이내로 끝납니다. Quby의 데이터팀은 고객을 위한 새로운 ML 기반 기능을 더욱 빠르게 개발할 수 있게 되었습니다.
- **에너지 사용량 절감**: Quby는 Waste Checker 앱을 통해 개인 맞춤형 추천으로 절약한 에너지가 시간당 6,700백만 kW가 넘는다는 것을 발견했습니다.

자세히 알아보기

databricks

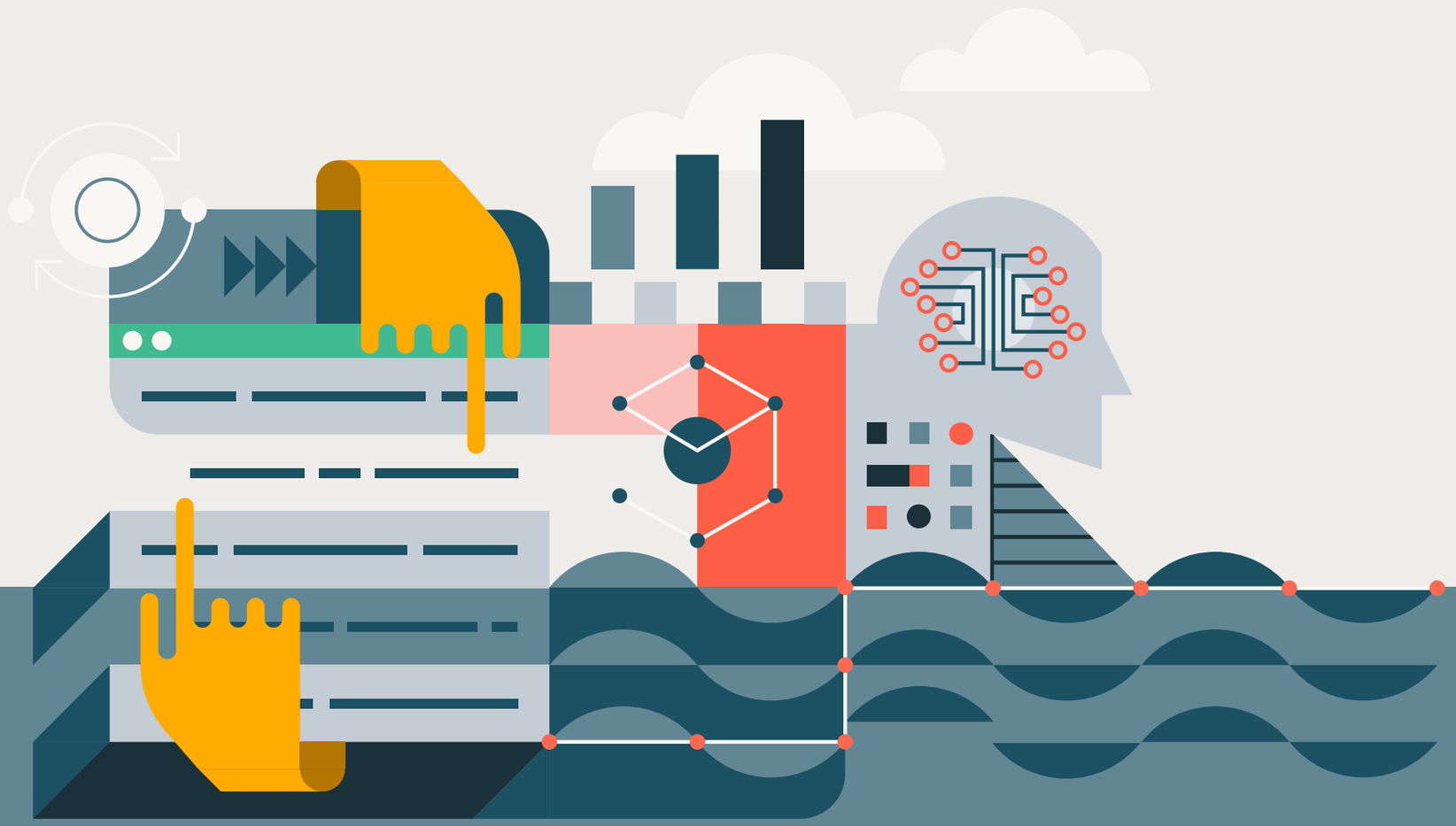
Databricks 소개

Databricks는 데이터 및 AI 회사입니다. Comcast, Condé Nast, Nationwide, H&M을 비롯하여 전 세계적으로 수천 개의 고객사가 Databricks의 개방적인 통합 플랫폼을 데이터 엔지니어링, 머신 러닝, 분석에 사용합니다. Databricks는 샌프란시스코에 본사가 있으며 전 세계에 지사를 두고 있는 벤처입니다. Apache Spark™, Delta Lake 및 MLflow의 원 제작자가 설립한 Databricks는 데이터 팀이 세계의 어려운 문제들을 해결할 수 있도록 지원하는 사명이 있습니다. Databricks에 대한 자세한 정보를 확인하려면 팔로우하세요.

Twitter, LinkedIn, Facebook

맞춤형 데모 예약

무료 체험 신청



databricks